Proceedings of the IJCAI-03 Workshop on Issues in Designing Physical Agents for Dynamic Real-Time Environments: World modelling, planning, learning, and communicating

Ubbo Visser
Patrick Doherty
Gerhard Lakemeyer
Manuela Veloso
(editors)

Held on August 11, 2003 in conjunction with the
International Joint Conference on Artificial Intelligence
Acapulco, Mexico

.

IJCAI-03 Workshop on Issues in Designing Physical Agents for Dynamic Real-Time Environments: World modelling, planning, learning, and communicating

Held on August 11, 2003 in conjunction with the International Joint Conference on Artificial Intelligence Acapulco, Mexico

Recent developments in multiagent systems (MAS) have been promising by achieving autonomous, collaborative behavior between agents in various environments. However, most of the agents, both software agents and physical agents, still have problems if the environment is dynamic and the agents have to act in real time. Examples are obstacle avoidance with moving obstacles or world models which are composed from egocentric views of numerous agents. Another aspect is the need for quick responses. In an environment where a number of agents build a team and both single agent decisions and team collaborative decisions have to be made methods have to be fast and precise. This workshop addresses various problems that occur with respect to these issues.

The main focus of this workshop will be methods from various areas such as world modelling, planning, learning, and communicating with agents in dynamic and real-time environments. Within this general theme we aim to bring together researchers to discuss the following topics:

- World modelling (quantitative, qualitative)
- Coaching (one agent gives advice to a group of agents)
- Planning with resources (especially time)
- Cooperation between agents (robot and/or humans)
- Communication between agents (implicit, non-verbal, or verbal one)
- Real-time systems software issues (often ignored but important if serious about real-time issues in robotics)
- Scalability and robotics interfacing issues (demands a great deal of support from the initial design of the system)

In the last decade, a lot of effort has been invested to develop methods that can be used with multi-agent systems. The language development in the area of communication between agents (ACL) might act as the first example. Speech acts serve as the basic principle and various protocols have been invented (e.g. auctions, contract-nets, etc.). Can we transfer these results to environments where quick decisions have to be made? Consider planning as another example: there are promising methods for path planning, but do they still hold if the observed obstacles are moving? Learning is another example: we need on-line learning in a real-time scenario to give agents the option to learn more about their environment. Usually, learning takes a fair amount of time but sometimes this time is not available. Can we find methods which will consider these restrictions?

This workshop addresses researchers from various areas in AI who want to discuss the mentioned issues from their point of view. How can we develop new methods or adapt existing methods to meet these demands?

We received 18 submissions for the workshop. 13 contributions have been selected for oral presentation at the workshop. These proceedings contain all papers, which can be roughly categorized with the help of the following sketch of a general multiagent architecture. Please note that this is only a rough categorization and that there are a number of papers that belong to more than one component.

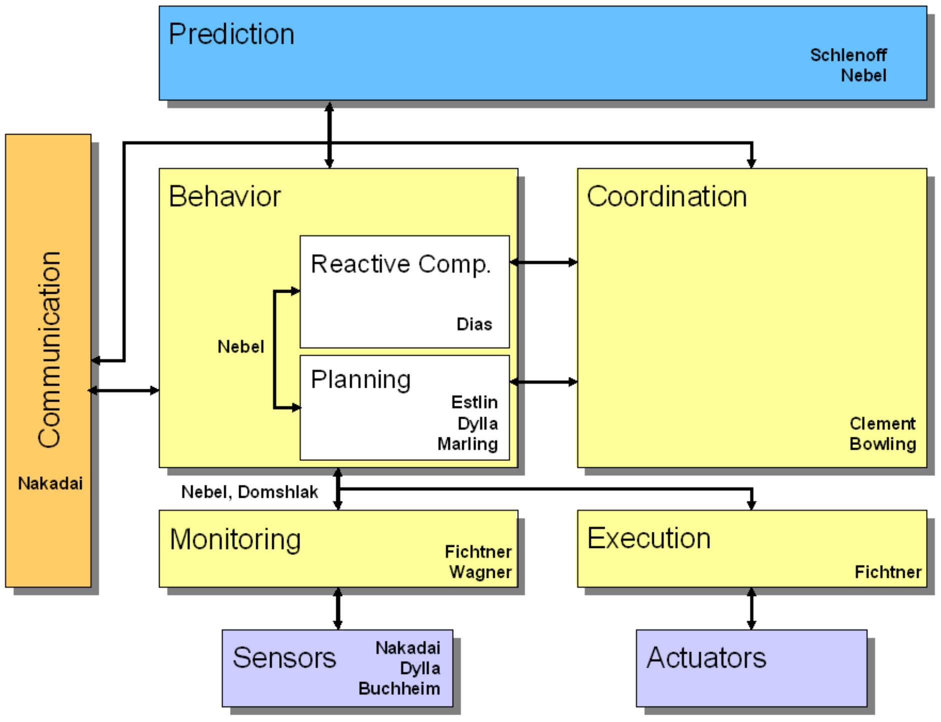


Figure 1: A general multiagent systems architecture. The papers are roughly categorized

The majority of contributions touch the areas of behavior modelling, which we divided into **planning** and **reactive components**. *Estlin et al.* represent the former and describe the OASIS system, which utilizes techniques from learning and planning for Mars missions. These missions aim long distance travelling. *Dylla et al.* show how plans can be specified with IPC-GOLOG. The authors developed this language as a variant of the logic-based language GOLOG and demonstrate how agents can be modelled for a robotic soccer scenario. *Marling et al.* deal with a case-based reasoning approach for both planning and world-modelling. This approach is used to help physical robotic soccer agents to plan individual moves and strategies. *Dias et al.* represent a reactive behavior approach. Their hypothesis is that a large control system (IDEA) can be structured as a collection of interacting control agents, each organized around the same fundamental structure. *Nebel & Babovich* propose goal-converging behavior networks and self-solving planning domains and deal with both, planning and reactive components.

Their paper can also be categorized between the **behavior** and **monitoring** and **execution**. The same holds for *Domshlak & Lawton's* contribution that describes an approach on planning for multiagent execution. They introduce a model of planning and execution that treats qualitative and quantitative effects of agents differently. The paper of *Wagner et al.* deals with monitoring only. The authors claim that qualitative spatial knowledge representation can help to overcome obstacles in highly dynamic environments. Their approach is based on an extended panorama. *Fichtner et al.* propose an approach for intelligent execution monitoring. A logical world representation allows planning and reasoning about world states.

Two papers especially deal with multiagent **coordination**. *Clement et al.* describe a Shared Activity Coordination (SHAC) framework, which provides a decentralized algorithm for negotiating the scheduling of shared activities. *Bowling et al.* introduce the concept of a *play* as a team plan and combine both reactive and deliberative principles.

Two papers can be categorized in the area of **prediction**. Autonomous vehicle scenarios are one example where the prediction of objects (obstacles) is crucial. *Schlenoff et al.* discuss an approach to predict location of moving objects during on-road navigation. The contribution of *Nebel & Babovich* can also be categorized in this group. They argue that behavior networks converge to given goals regardless of the particular action scheme.

Nakadai et al. represent the **communication** component. Their problem is a situation where a humanoid robot has to deal with a number of simultaneous talkers. Their approach follows a speech recognition by audio-visual integration. This paper also represents the **sensor** component of our general architecture. In addition, the contribution by *Dylla et al.* also deals with sensor data and can therefore be categorized in this component. *Buchheim et al.* propose a flexible and generic software framework for world modelling enabling a realization of selective attention mechanisms.

Program Committee and Reviewers

We are grateful to the following members of the international program committee for helping us to make this a high quality workshop:

- Minoru Asada, Osaka University, Osaka, JAPAN
- Andreas Birk, International University, Bremen, GERMANY
- Hans-Dieter Burkhard, Humboldt Universität, Berlin, GERMANY
- Gregory Dudek, McGill University, Montreal, CANADA
- Dieter Fox, University of Washington, Seattle, WA, USA
- Christian Freksa, Universität Bremen, Bremen, GERMANY
- Uli Furbach, Universität Koblenz-Landau, Koblenz, GERMANY
- Otthein Herzog, Universität Bremen, Bremen, GERMANY
- Sven König, Georgia Institute of Technology, Atlanta, GA, USA
- Paul Levi, Universität Stuttgart, Stuttgart, GERMANY
- Elena Messina, National Institute of Standards & Technology (NIST), Gaithersburg, MD, USA
- Daniele Nardi, Università di Roma “La Sapienza”, Rome, ITALY
- Bernhard Nebel, Universität Freiburg, GERMANY
- Rolf Pfeifer, Universität Zürich, Zürich, SWITZERLAND
- Thomas Röfer, Universität Bremen, Bremen, GERMANY
- Andrzej Skowron, Warsaw University, Warsaw, POLAND
- Peter Stone, The University of Texas at Austin, Austin, TX, USA

Contents

Learning and Planning for Mars Rover Science

Tara Estlin, Rebecca Castano, Robert Anderson, Daniel Gaines,
Forest Fisher, and Michele Judd
Jet Propulsion Laboratory
California Institute of Technology
4800 Oak Grove Dr., Pasadena CA 91109
Tara.Estlin@jpl.nasa.gov

Abstract

With each new rover mission to Mars, rovers are traveling significantly longer distances. In some cases, distances are increasing by orders of magnitude from previous missions. This increase enables not only the collection of more science data, but causes a large rise in the number of new and different science collection opportunities. In this paper, we describe the OASIS system, which provides autonomous capabilities for dynamically pursuing these science-collection opportunities during long-range rover traverses. OASIS utilizes techniques from both machine learning and planning and scheduling to address this goal. Machine learning techniques are applied to analyze data as it is collected and quickly determine new science tasks and priorities on these tasks. Planning and scheduling techniques are used to alter the rover's behavior so new science measurements can be performed while still obeying resource and other mission constraints. In addition to describing our system, we also discuss how we are testing OASIS, including the use of Mars rover prototypes and validation using data gathered from expert planetary geologists.

1 Introduction

As planetary exploration continues to increase, the use of robotic vehicles to explore and analyze planet surfaces will also expand. The Mars Pathfinder mission not only demonstrated the feasibility of sending rovers to other planets, but displayed the significance of such missions to the scientific community. The Mars Exploration Rovers (MER) mission is set to launch this year, and will send two new rovers to the Martian surface. Furthermore, additional rover missions are already planned to the red planet, which will provide major leaps in smart, surface laboratory measurements. With each new mission, rovers are able to travel significantly longer distances and collect increasing amounts of valuable science data. However, they must perform this task in unknown environments where unexpected conditions can easily be encountered. The Pathfinder rover traveled approximately 100 meters during its 90 day lifetime [Mishkin, *et al.*, 1988]. In contrast, the MER rovers will travel up to 100 meters per day, and future missions will likely continue to extend this measure. Though longer-range traverses enable rovers to explore new territory and collect large volumes of data, they also place increasing demands on operating these missions. Collected images and other science data must be analyzed (typically on earth), and this process must be performed quickly if that data is used to direct additional science measurements by the rover. Furthermore, rover operations for both Pathfinder and MER are handled by manually creating sequences of rover commands on the ground and then uploading them to the rover. This process is very time- and labor-intensive and does not allow for the dynamic adjustment of rover behavior if anything unexpected occurs, including faults and new science opportunities.

This paper describes the Onboard Autonomous Science Investigation System (OASIS) [Castano, *et al.*, 2003], which is directed at providing autonomous capabilities for rover science operations during long-range traverses. In upcoming missions, rovers will traverse many kilometers between pre-designated science sites. OASIS was developed to support science data analysis and new science collection during these long traverses. OASIS consists of several modules, including: 1) a data analysis system that uses machine learning techniques to analyze collected data and produce new science collection goals and 2) a planning and scheduling system that dynamically incorporates new science goals into the current rover command sequence and interacts with the onboard control software to achieve this goal. This system is currently being tested on data collected during test operations for the MER mission as well as with the Rocky 8 rover, a research rover built and supported at JPL.

Science data analysis in OASIS is performed using several different machine-learning techniques, which can prioritize acquired science data for downlink to earth and create new science goals for the rover to achieve. This paper concentrates on the latter capability of creating new science goals. More information on prioritizing data for downlink can be found in [Castano, *et al.*, 2003]. Three different prioritization methods have been devel-

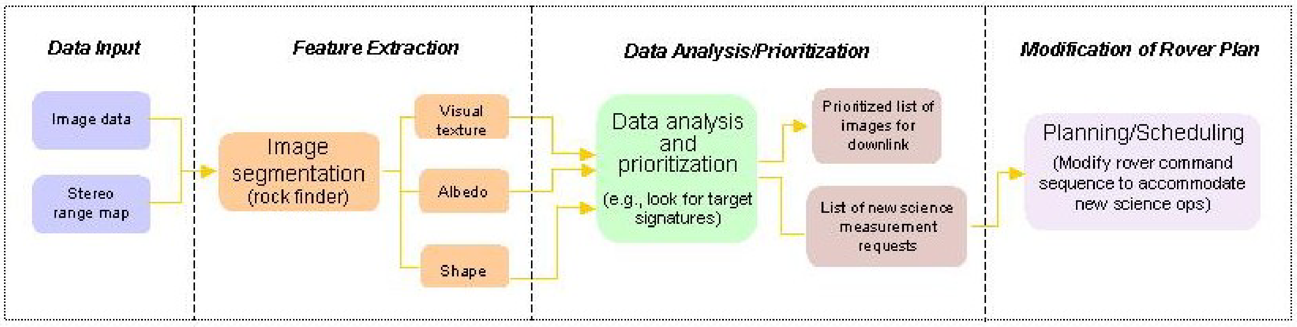

Data Input
Feature Extraction
Data Analysis/Prioritization
Modification of Rover Plan
Image data
Stereo range map
Image segmentation (rock finder)
Visual texture
Albedo
Shape
Data analysis and prioritization
(e.g., look for target signatures)
Prioritized list of images for downlink
List of new science measurement requests
Planning/Scheduling
(Modify rover command sequence to accommodate new science ops)

Figure 1. Overview of OASIS system architecture. OASIS consists of three major components: Feature Extraction, Data Analysis/ Prioritization, and Planning and Scheduling.

oped for OASIS. All use extracted rock features to rank rocks in terms of scientific importance. The first technique, *target signature detection*, recognizes pre-specified signatures that have been identified by ground scientists as data of high interest. The second technique, *novelty detection*, identifies unusual signatures that do not conform to the statistical norm for the region. The last technique, *representative sampling*, prioritizes science measurements by ensuring data is collected on representative rocks of the traversed region. These three prioritization methods are used to trigger opportunistic science observations by identifying valuable new science opportunities that, if possible, should be taken advantage of during the rover's traverse.

When science opportunities arise on a traverse, a planning and scheduling system is used to determine the necessary rover activities to achieve the new science goals. Based on an input set of prioritized goals and the rover's current command sequence, the planner generates a modified sequence of activities that satisfies as many new goals as possible while still preserving high-priority activities already in the sequence and obeying resource and other operation constraints (e.g., such as ensuring there will be enough power to complete the day's activities). Our planner uses a *continuous planning* approach, where plans are dynamically modified in response to changing events and goal information. In this approach, the planner continually monitors the execution of commands on the rover and information on resource utilization and current states. It also accepts new science goals as they become available. As information is acquired regarding these items, the planner updates its version of the plan. From these updates, new problems and/or opportunities may arise, requiring the planner to re-plan in order to accommodate the unexpected events.

The remainder of this paper is organized as follows. We begin by presenting the OASIS system, including characterizing the full architecture and presenting a more detailed explanation of the system components. We will then describe our testing plan for OASIS, which includes using Martian data from upcoming missions, as well as robotic vehicles developed at JPL. Finally, we will discuss related work and present our conclusions.

2 OASIS System

The OASIS system architecture is shown in Figure 1. As highlighted in the figure, OASIS is comprised of three major components:

- **Feature Extraction**: Enables extraction of features of interest from collected images of the surrounding terrain. This component both locates rocks in these images and extracts rock properties, such as shape and texture.
- **Data Analysis/Prioritization**: Uses extracted features to assess the scientific value of the planetary scene and to generate new science objectives that will further contribute to this assessment. This component consists of three different prioritization algorithms, that analyze collected data, prioritize identified rocks, and generate a new set of observation goals to gather further data on rocks which were ranked high priority.
- **Planning and Scheduling**: Enables dynamic modification of the current rover command sequence (or plan) to accommodate new science requests from the data analysis unit. This component uses a *continuous planning* approach to iteratively adjust the plan as new goals and/or faults occur.

OASIS operates in an autonomous fashion where the data analysis system can be seen as driving new science exploration. First, new science data is received by the Feature Extraction component. Currently, we have focused the system on analyzing rocks within \image data, but plan to expand to other types of data, such as spectrometer measurements. Images are broken down by first locating individual rocks in each received image, and second, by extracting a set of rock properties (or features) from each identified rock. Extracted rock properties are then passed to the Data Analysis component of the system.

This component consists of three different prioritization algorithms, which analyze the data by searching for items such as pre-known signatures of interest, which have been identified by scientists on earth, or novel rocks (i.e., outliers) that have not been seen in past traverses.

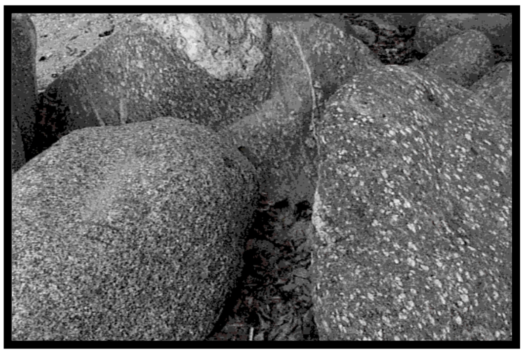
As shown in Figure 1, this analysis produces two main products. One is a set of prioritized images for transmission to Earth. Currently, spacecraft, such as rovers, can collect significantly more data than can be transmitted to Earth due to communication limits. OASIS ranks images by scientific importance so more valuable images get transmitted first for further analysis on the ground. This paper is focused on the second product, which is a list of new science measurement requests. OASIS uses the output of its three prioritization algorithms to dynamically produce a list of new science measurements that will take advantage of new and interesting data collection opportunities. In current rover missions, images and other science measurements are only sent to earth once or twice during the day. Furthermore, many images cannot be sent at all due to the communication restrictions mentioned above. This setup means that many valuable science opportunities may be lost. One problem is that by the time images are sent from Mars to Earth, analyzed on the ground by scientists, and a new set of measurement requests determined and sent back to Mars, the rover will likely have passed the object of interest. Another problem is the opportunity may never be recognized if the identifying data is never sent to Earth for analysis. By analyzing data onboard, OASIS enables these new science opportunities to be dynamically realized.

New science measurement requests (or goals) are passed to the planning and scheduling module, which produces a modified set of actions in order to achieve as many new science goals as possible, without violating resource or other mission constraints. In current mission operations, rover behavior is directed by manually hardcoding sequences of commands on Earth and then uploading these sequences to the rover. Sequence changes are rarely performed onboard and if something unexpected happens, the rover must contact earth for further instructions. The planning and scheduling component addresses this problem by using a model of rover operations and constraints to dynamically modify the current rover plan in order to accommodate new science goals. This component can also monitor plan execution and continue to modify the rover command plan if other unexpected events or faults occur.

Next, we discuss each of the OASIS components in more detail.

2.1 Feature Extraction

The first step in the OASIS system is analyzing rock features from images taken onboard the rover. As the rover traverses, it takes a series of images to support not only science, but also navigation operations. Images may be taken from several different cameras to capture information on the surrounding terrain for science analysis and/or



(a)

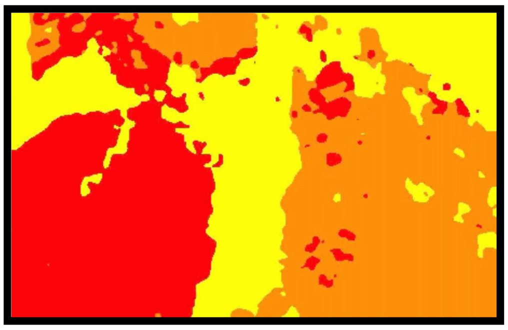

(b)

Figure 2. Examples of visual texture providing information about the geologic texture of rocks. (a) original image (b) image segmented based on texture.

assist in path planning, obstacle avoidance, etc. Our initial emphasis in OASIS has focused on image analysis and the characterization of surface rocks. Rocks are among the primary features populating the Martian landscape and the understanding of rocks on the surface is a first step leading towards more complex regional geological assessments by a robotic vehicle.

Rocks are located in the images by determining the ground plane from stereo range data, and then producing a height image and level contours for that image. These contours can be connected from peaks to the ground plane to identify rocks in each image [Gor, *et al.*, 2001].

Next, a set of properties is extracted from each rock. Our feature extraction priorities are based upon our knowledge of how a geologist in the field would extract information. Important features to look for and categorize include albedo (an indicator of rock surface reflectance properties), visual texture (which provides valuable clues to mineral composition and geological history), shape, size, color and arrangement of rocks. Currently our system identifies the first three of this set; future work will expand this to cover additional features. Each property or feature is measured using a different technique [Gilmore, *et al.*, 2000; Castano *et al.*, 2002]. For instance, visual texture is measured by computing gray-scale intensity variations at different orientations and spatial frequencies within the image. Figure 2 shows visual texture information produced from one sample image.

2.2 Data Analysis and Prioritization

The second step in the OASIS system is to use the extracted features to prioritize rocks. Three prioritization techniques are used, which employ different machine learning methods. The results from this analysis are then used to identify rocks that should be further analyzed and produce a new set of science measurement goals.

Key Target Signature

The first prioritization technique, key target signature, enables scientists to efficiently and easily stipulate the value and importance of certain features. Scientists often have an idea of what they expect to find during a rover mission and/or are looking for specific clues that reflect signs of life or water (past or present). Using this technique, target feature vectors can be pre-specified and an importance value assigned to each of the features. Rocks are then prioritized as a function of the weighted Euclidean distance of their extracted features from the target feature vector.

Novelty Detection

The second prioritization technique, novelty detection, detects and prioritizes unusual rocks that are dissimilar to previous rocks encountered. We have looked at three different learning techniques for novelty detection. First, we have developed a distance-based k-means clustering approach, in which a set of rocks are clustered and any new rock that is a great distance from any of the cluster centers is considered novel. In the second method, the probability density over the feature space for a set of rocks is approximated using a Gaussian mixture model. The novelty of a new rock is inversely proportional to the probability of that rock being generated from the learned mixture model. The third method uses a discrimination based kernel one-class classifier approach. We treat all previous rock data as the "positive class" and learn the discriminant boundary around that class. Future rocks with features falling outside the boundary are considered novel. An example of detecting a novel rock using data collected from rover field tests is shown in Figure 3.

These three approaches represent the three dominant flavors of machine learning techniques for novelty detection: distance-based, probability-based (i.e., generative), and discriminative. Considering all three types in one hybrid approach allows us to tradeoff their respective advantages and disadvantages.

Representative Sampling

In order to understand the region being traversed, it is important to have information on representative rocks, vs. very interesting or unusual rocks. A region is likely to be populated by several types of rocks, with each type having a different abundance. Thus, a uniform sampling will be biased towards the dominant class of rock present and may result in smaller rock classes not being represented at all in the downlinked data.

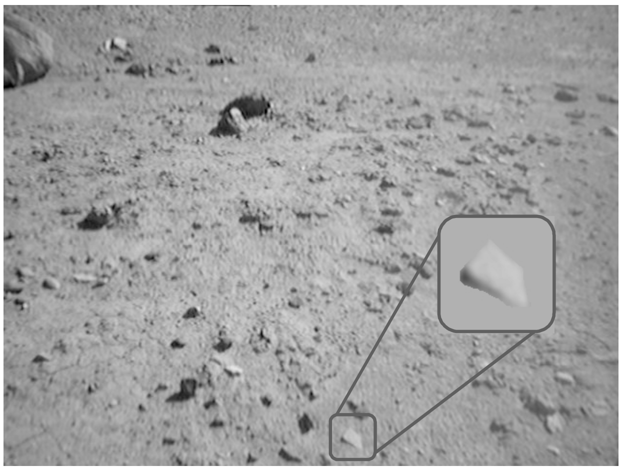


Figure 3: Detection of significant novel rock. The marked rock is a piece of petrified wood that was discovered during rover field tests for the MER mission. This piece of wood was identified as novel by the OASIS system, however was not identified by the remotely located geologists during the rover tests.

The third prioritization technique, representative sampling, provides an understanding of the typical characteristics of a region. Rocks are clustered into groups with similar properties and the data is then prioritized to ensure that representative rocks from each class are sampled. To determine the classes, the rock property values are connected together in a series to form a feature vector. A weight is assigned to the importance of each property. Unsupervised clustering is then used to separate the feature vectors into similar classes. We currently use a k-means clustering technique due to its relatively low computational requirements. However, other unsupervised methods could also be employed. For each class of rocks, this technique can find the most representative rock in the class (i.e., the single rock in any image that is closest to the mean of the set) or rank rocks according to this metric.

Science Alert

Using the above determined priorities, the data analysis software can then flag rocks that should be further analyzed and produce a new set of measurement goals to further characterize the identified rocks. We call this capability *science alert*, since it alerts other onboard software that new and high priority science opportunities have been detected. The number of new goals produced by the data analysis software will vary depending on the constraints of the mission. Some missions may want only limited science alert capabilities, and thus new opportunities would only be flagged it they were deemed critical. Other missions may allow onboard analysis to direct a larger portion of planned science measurements.

Science alert may also involve several different levels of reaction. OASIS has been designed so a spectrum of reactions can be supported. The most basic reaction is to adjust the rover plan so that the flagged data is immediately sent back to Earth for further analysis and the rover holds at the current position, delaying other tasks. The next step would likely be to collect additional data at the current site before transmitting data to earth. Further steps include having the rover alter its path to get closer to objects of interest before taking additional measurements and/or scheduling a close contact measurement (such as with a microscopic imager). These operations would provide new data that could not be obtained through image analysis alone. The level of allowed reaction will likely be determined by the constraints and goals of the rover mission. Reaction capabilities may also be allowed to vary over the course of the mission.

2.3 Planning and Scheduling

Once the data analysis software has identified a set of new science targets, these targets are passed to onboard planning and scheduling software that can dynamically modify the current rover plan in order to collect the new science data. This component takes as input the new set of science requests, the current rover command sequence (or plan), and a model of rover operations and constraints. It then evaluates what new science tasks could be added to the current plan while ensuring other critical activities are preserved and no operation or resource constraints are violated.

CASPER Planner

Planning and scheduling capabilities are provided in OASIS by the Continuous Activity Scheduling, Planning and Re-Planning (CASPER) system [Estlin, *et al*., 2002; Chien, *et al*., 2000]. CASPER provides a generic planning and scheduling application framework that can be tailored to specific domains. Its components include:

- An expressive modeling language to allow the user to naturally define the application domain.
- A constraint management system for representing and maintaining domain operability and resource constraints.
- A set of search strategies and repair heuristics.
- A temporal reasoning system for expressing and maintaining temporal constraints.
- A graphical interface for visualizing plans.
- A real-time system that monitors plan execution and modifies the current plan based on activity, goal, state and resource updates.

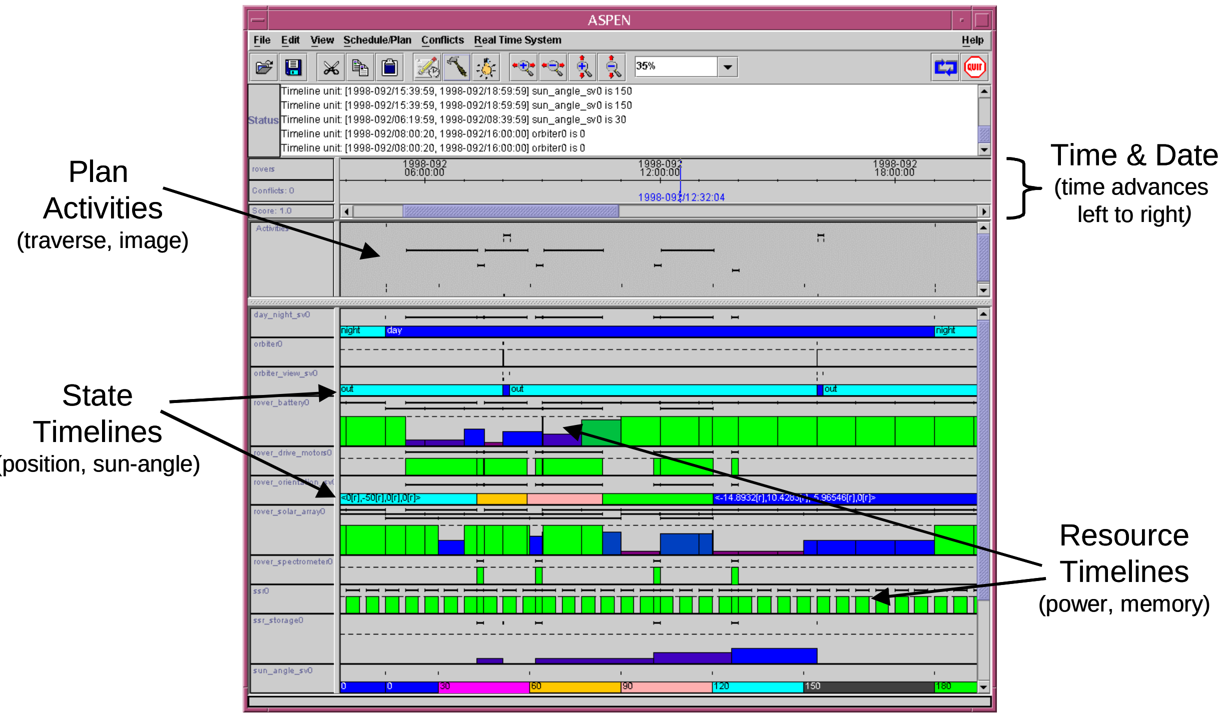
CASPER employs a *continuous planning* technique where the planner continually evaluates the current plan and modifies it when necessary based on new state and resource information. Rather then consider planning a batch process, where planning is performed once for a certain time period and set of goals, the planner has a current goal set, a current rover state, and state projections into the future for that plan. At any time an incremental update to the goals or current state may update the current plan. This update may be an unexpected event (such as a new science opportunity) or a current reading for a particular resource level (such as power). The planner is then responsible for maintaining a plan consistent with the most current information. And since things rarely go as expected, especially during planetary surface operations, the planner stands ready to continually modify the plan.

A plan consists of a set of grounded (i.e., time-tagged) activities that represent different rover actions and behaviors. Activities can be at different levels of abstraction, where low-level activities typically correspond to direct rover commands. For example, a plan typically contains several *traverse* activities that move the rover between different locations in order to visit science targets. Rover state in CASPER is modeled by a set of plan timelines, which contain information on both states, such as rover position, and resources, such as power. Timelines are calculated by reasoning about activity effects and represent the past, current and expected state of the rover over time. As time progresses, the actual state of the rover drifts from the state expected by the timelines, reflecting changes in the world. State updates are relayed back from sensors and the rover control software. As these updates are received, CASPER updates the relevant timeline models with actual state values, resource values, activity completion times, etc. Each of these updates may introduce problems into the current plan, such as a power over-subscription due to a long traverse or an instrument being in the incorrect position to perform a particular science reading. These problems (or plan conflicts) cause CASPER to perform plan modifications to bring the plan back into sync with the current state and set of goals. An example of a plan in the CASPER GUI is shown in Figure 4.

To produce and/or modify a current rover plan, CASPER uses an iterative repair algorithm [Zweben *et al*., 1994], which classifies plan conflicts and attacks them individually. Conflicts occur when a plan constraint has been violated where this constraint could be temporal or involve a resource, state or activity parameter. Conflicts are resolves by using one or more plan modifications such as moving, adding, or deleting activities. One example of a conflict is when a new science activity oversubscribes a resource such as power or memory. Possible resolutions to this conflict might be moving the science activity to a part of the plan that doesn't oversubscribe that resource, deleting the science activity, or moving and/or deleting other contributing activities.

Path Planning

To provide spatial reasoning capabilities to CASPER, we are using a global path-planning module, which provides rover route information to the planner based on a map of the



```
Timeline unit: [1998-092/15:39:59, 1998-092/18:59:59] sun_angle_sv0 is 150
Timeline unit: [1998-092/15:39:59, 1998-092/18:59:59] sun_angle_sv0 is 150
Timeline unit: [1998-092/06:19:59, 1998-092/08:39:59] sun_angle_sv0 is 30
Timeline unit: [1998-092/08:00:20, 1998-092/16:00:00] orbiter0 is 0
Timeline unit: [1998-092/08:00:20, 1998-092/16:00:00] orbiter0 is 0
```

Figure 4: Sample rover plan displayed in planner GUI. Plan activities are shown as bars in upper portion of window, where bars represent the start and end time of each activity. State and resource timelines are shown in bottom portion of screen and show the effects of the plan as time progresses. Time is depicted as advancing from left to right.

rover's surrounding environment. This module is intended to give a global perspective of the rover's anticipated path as opposed to the local perspective that would be considered by obstacle avoidance software. We are assuming that for most rover operations some global map information would be available through orbital or descent imagery, or from panoramic imagery generated onboard the rover itself. We are also assuming this map information map be incomplete and certain terrain features and/or obstacles may be missing.

CASPER queries the path planner for two main pieces of information. The first piece is estimated distances between science target or other designated traverse waypoints. The second piece is a list of intermediate-waypoint coordinates that can be used to direct the rover's traverse to a particular targets. Path-distance information is used by the planner to estimate duration and power required for rover traverses between targets. Intermediate waypoints are used to track the rover's progress during a traverse. To provide path planning information to our system, we are currently using the D* path planner, which produces paths in partially known or changing environments using an efficient and optimal algorithm [Stentz, 1994].

3 System Testing

We are in the process of testing the OASIS system using data gathered during rover field tests for upcoming missions as well as using several JPL research rovers in the JPL Mars Yard.

The data analysis component is currently being tested using a suite of image data collected during rover field experiments performed in Flagstaff, AZ. (These field experiments were done in preparation for the upcoming 2003 MER rover mission.) One of the primary goals of using this data to test OASIS is to not only test our system on realistic data, but to also ensure that the prioritizations our algorithms produce are comparable to those made by planetary geologists. Our approach for testing is to gather sample prioritizations from expert planetary geologists on various collections of images. Expert rankings are input using a web-based application that enable experts from across the country to easily prioritize images and add explanations for their decisions. We are using statistical methods to combine these expert prioritizations and compare them with the prioritizations produced by our algorithms.

The planning component has already been used in several tests [Estlin, *et al.*, 2002] using two JPL rovers, Rocky 7 and Rocky 8, which are pictured in Figure 5.

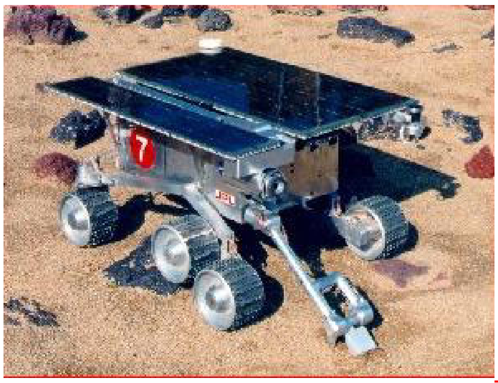
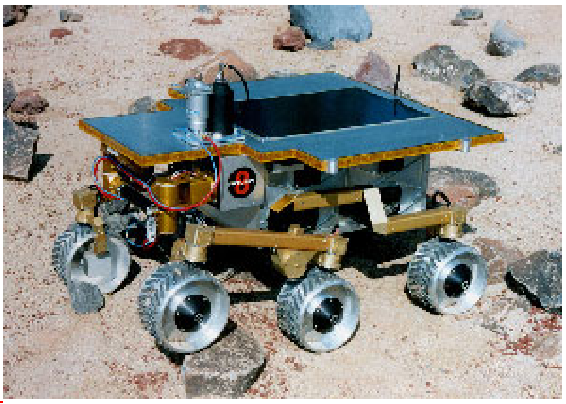



Figure 5: JPL Rocky 7 and Rocky 8 rovers

Rocky 7 is approximately the same size and mass as the 1997 Mars Pathfinder rover, Sojourner. It employs a rocker-bogie six-wheel configuration, and is a partially-steered vehicle, where it only has steering capability on two corners. In contrast, Rocky 8 is roughly an order of magnitude larger than Rocky 7 and is similar in size to the twin MER rovers, set to launch later this year. Rocky 8 also employs a rocker-bogie six-wheel configuration, however it is a fully-steered vehicle with all-wheel drive and all-wheel steering.

The planner was used to produce an initial rover plan based on a set of science objectives (e.g., perform an image at location A, perform a spectrometer read at location B, etc.) and to dynamically modify that plan when unexpected events occurred during execution (e.g., more power was required for a traverse or science activity than originally estimated). Tests were performed in the JPL Mars Yard. The initial plan contained 53 different activity instances and took the planner 3.7 seconds to construct. The planning horizon for these tests was 4 hours. Replans took an average of 7 seconds.[1] During these tests the planner interacted with the rover control software in several different ways. One, it dispatched commands from the plan for execution. Two, it monitored the success or failure of these commands. And three, it monitored a set of resource and state information including items such as rover position, power levels, and onboard memory levels. If unexpected events occurred, then the plan was dynamically revised to accommodate the new information. Note, that in these early tests all unexpected events were undesirable (e.g., resources oversubscribed, traverses taking longer than estimated), and none corresponded to new science opportunities.

The above-mentioned tests will be significantly expanded on this year, including 1) testing all components using real rovers 2) testing the online incorporation of new science goals, and 3) testing with additional real data sets gathered during rover traverses both on earth and Mars.

[1] Performance numbers reported in this paper were run on a Linux 1.7 GHz Pentium 4 workstation.

4 Related Work

The idea of having a scientific discovery system direct future experiments is present in a number of other systems. Work on learning by experimentation, such as IDS [Nordhausen and Langley, 1993] and ADEPT [Rajamoney, 1990], varied certain quantitative and qualitative values in the domain and then measured the effects of these changes. OASIS differs from these systems in that it interacts with the environment to perform experimentation and it is specialized to particular problems and scenarios in planetary science. OASIS is also integrated with a planning system, which constructs the detailed activity sequence needed to perform new science experiments.

Other work has used experimentation to learn from the environment but experiments again have not been scientifically driven. EXPO [Gil, 1993] integrates planning and learning methods to acquire new information by interacting with an external environment. However, while EXPO tries to improve its planning-related domain knowledge, OASIS learns prioritization models of geological terrain features. Another example of learning about the environment is the Minerva museum tour-guide robot [Thrun, et al., 1999]. Minerva learns several pieces of environment knowledge, including maps, sensor models and travel times between museum locations. Minerva's reactive planner uses learned travel times to dynamically alter its tour of the museum based on time limits. Conversely, OASIS learns new prioritized science goals for the rover to achieve, and uses an activity planner/scheduler that reasons about mission goals and resource and state constraints.

Several researchers have addressed methods for extracting features from data with the intention of performing the operations onboard a spacecraft. Gulick *et al.* [2001] presented methods for locating rocks in an image using information about the sun angle, identifying the horizon and recognizing layers. Gazis and Bishop [Gazis and Bishop, 2002] and Ramsey et al [Ramsey, *et al.*, 2002] have both looked at analyzing point spectra for mineral detection. There has also been work on developing a framework for feature extraction and event detection onboard Earth orbiting satellites (EVE) [Tanner, *et al.*, 2001]. Our work has specifically focused on identifying and analyzing rocks in gray-scale images thus far and, in contrast to the work mentioned here, takes the next step of using the feature extraction to determine desirable additional actions a rover could autonomously take.

A number of other systems have used planning methods to coordinate robot behavior. [Gat, 1992; Bonasso, *et al.*, 1997; Alami, *et al.*, 1998]. However, these systems generate plans in a batch approach where plans are generated for a certain time period and if re-planning is required, an entire new plan must be produced. In OASIS, plans are continuously modified in response to changing conditions and goals. The CPS planner, which is also directed towards rover operations, generates contingent plans, which are then executed onboard and can be modified at certain points if failures occur [Bresina, *et al.*, 1999]. This planner takes a more limited approach then the OASIS planner, since the only plan modifications that can be performed during execution are those that have previously identified as possible change points. Furthermore, none of these systems has been integrated with a machine learning system that drives future plan goals.

5 Conclusions

This paper presents the OASIS system, which is being developed to support autonomous science operation during long-range rover traverses. OASIS integrates techniques from machine learning with planning and scheduling to dynamically analyze science data, request new science operations, and generate a new plan of action to support those requests. Often, in current rover missions, volumes of data are collected during rover traverses, however much of this data cannot be sent back to earth due to communication restrictions. OASIS enables this data to be analyzed onboard the rover and then used to determine new science measurement goals for objects of high interest. This system is currently being tested using several real rovers and using data gathered during rover field experiments.

Acknowledgments

The research described in this paper was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

References

[Alami *et al.*, 1998] R. Alami, R. Chautila, S. Fleury, M. Ghallab, and F. Ingrand. An Architecture for Autonomy. *International Journal of Robotics Research*. 17(4) April, 1998.

[Bonasso *et al.*, 1997] R. Bonasso, R. Firby, E. Gat, D. Kortenkamp, D. Miller, and M. Slack. Experiences with an Architecture for Intelligent, Reactive Agents. *Journal of Experimental and Theoretical Artificial Intelligence Research*, 9(1), 1997.

[Bresina *et al.*, 1999] J. Bresina, K. Golden, D. Smith, D., and R. Washington. Increased Flexibility and Robustness of Mars Rovers. *Proceedings of the International Symposium, on AI, Robotics and Automation for Space*. Noordwijk, The Netherlands, June 1999.

[Castano *et al.*, 2002] R. Castano, R.C. Anderson, J. Fox, J.M. Dohm, A.F.C. Haldemann, and W. Fink. Automating shape analysis of rocks on Mars. *Proceedings of the Lunar and Planetary Science Conference*, March 2002.

[Castano *et al.*, 2003] Rebecca Castano, Robert Anderson, Tara Estlin, Dennis Decoste, Forest Fisher, Daniel Gaines, Dominic Mazzoni, and Michele Judd. Rover Traverse Science for Increased Mission Science Return. *Proceedings of the 2003 IEEE Aerospace Conference.* Big Sky, Montana, March 2003.

[Chien, *et al.*, 2000] Steve Chien, Russell Knight, Andre Stechert, Rob Sherwood, and Gregg Rabideau. Using Iterative Repair to Improve the Responsiveness of Planning and Scheduling. *Proceedings of the Fifth International Conference on Artificial Intelligence Planning and Scheduling*, Breckenridge, CO, April 2000.

[Estlin, *et al.*, 2002] Tara Estlin, Forest Fisher, Daniel Gaines, Caroline Chouinard, Steve Schaffer, and Issa Nesnas. Continuous Planning and Execution for a Mars Rover. *Proceedings of the Third International NASA Workshop on Planning and Scheduling for Space*. Houston, TX, Oct 2002.

[Gat, 1992] Erann Gat. ESL: A Language for Supporting Robust Plan Execution in Embedded Autonomous Agents, *Proceedings of the Tenth National Conference on Artificial Intelligence*, San Jose, CA, July 1992.

[Gazis and Bishop, 2002] P. Gazis and J. Bishop. Development of rule-based autonomous spectral analysis techniques for planetary surfaces: preliminary results using lab spectra. *American Geophysical Meeting Fall meeting*, San Francisco, CA, Dec 2002.

[Gil, 1993] Yolanda Gil. Efficient domain-independent experimentation. *Proceedings of the Tenth International Conference on Machine Learning.* 1993

[Gilmore, *et al*., 2000] M. Gilmore, R. Castano, T. Mann, R. C. Anderson, E. Mjolsness, R. Manduchi, and R. S. Saunders. Strategies for autonomous rovers at Mars. in *J. of Geophysical Res*., Vol. 105, No. E12, Dec. 2000 pp. 29223-29237.

[Gor, *et al*., 2001] V. Gor, R. Castano, R. Manduchi, R. Anderson, and E. Mjolsness. Autonomous Rock Detection for Mars Terrain. *Space 2001, American Institute of Aeronautics and Astronautics*, Aug. 2001.

[Gulick, *et al*., 2001] V. Gulick, R. Morris, M. Ruzon, and T. Roush. Autonomous image analyses during the 1999 Marsokhod rover field test. *Journal of Geophysical Research-Planets*, 106 (E4): 7745-7763 Apr. 2001.

[Mishkin, *et al*., 1998] A. Mishkin, J. Morrison, T. Nguyen, H. Stone, B. Cooper, B. Wilcox. Experiences with Operations and Autonomy of the Mars Pathfinder Microrover. *Proceedings of the 1998 IEEE Aerospace Conference*. Aspen, CO, March 1998.

[Nordhausen and Langley, 1993]. B. Nordhausen and P. Langley. An integrated framework for empirical discovery. *Machine Learning*. 12:17-47.

[Ramsey, *et al*., 2002] J. Ramsey, P. Gazis, T. Roush, P. Spirtes, and C. Glymour. Automated remote sensing with near infrared reflectance spectra: carbonate detection. *American Geophysical Meeting Fall meeting*, San Francisco, CA, Dec. 2002.

[Rajamoney, 1990] S. Rajamoney, S. A computational approach to theory revision. In Shrager, J., and Langley, P., eds. *Computational Models of Scientific Discovery and Theory Formation*. San Mateo, CA: Morgan Kaufman. 225-254.

[Stentz, 1994] Anthony Stentz. Optimal and Efficient Path Planning for Partially-Known Environments. *Proceedings of the IEEE International Conference on Robotics and Automation*, San Diego, CA, May 1994.

[Tanner, *et al*., 2001] S. Tanner, K. Keiser, H. Conover, D. Hardin, S. Graves, K. Regner, R. Wohlman, R. Ramachandran, M. Smith. EVE: An on-orbit data mining testbed. *IJCAI-01 Workshop on Knowledge Discovery from Distributed, Heterogeneous, Autonomous, Dynamic Data and Knowledge Sources*. Seattle, Washington, Aug. 2001.

[Thrun, 1999] S. Thrun, M. Bennewitz, W. Burgard, A. Cremers, F. Dellaert, D. Fox, D. Hahnel, C. Rosenberg, N. Roy, J. Schulte, and D. Schulz. MINERVA: A Second-Generation Museum Tour-Guide Robot. *Proceedings of the IEEE International Conference on Robotics and Automation*, Detroit, MI May 1999.

[Zweben, *et al*., 1994] M. Zweben, B. Daun, E. Davis, and M. Deale. 1994. Scheduling and Rescheduling with Iterative Repair, In *Intelligent Scheduling*, Morgan Kaufmann, San Francisco, CA. 1994. 241-256.

Specifying Multirobot Coordination in ICPGolog — From Simulations towards Real Robots —

F. Dylla and **A. Ferrein** and **G. Lakemeyer**
Knowledge-based Systems Group, Ahornstr. 55, RWTH Aachen Germany
Phone: +49 241 80-21534, Fax: +49 241 80-22321
{dylla, ferrein, gerhard}@cs.rwth-aachen.de

Keywords: reasoning about actions and change, knowledge representation

Abstract

Deliberation in highly-dynamic domains such as robotic soccer requires a rich representation language that can deal with continuous change, uncertainty, and multiple agents, among other things. For this purpose we have developed the language ICPGOLOG, a variant of the logic-based action language GOLOG. We show how to specify plans for soccer agents such as playing a double pass in ICPGOLOG and report on experimental results in the ROBOCUP SIMULATION league. We have also recently integrated ICPGOLOG as the high-level control language into our MID-SIZE soccer team. We discuss the software architecture and some of the differences between agent modeling in the SIMULATION and MID-SIZE league.

1 Introduction

Highly dynamic real-time domains like robotic soccer place stringent requirements on the decision making process of agents. An action must be settled nearly immediately after new sensory information is received. There is little time to reason about the next action to perform. Especially in soccer, it is better to do something, even if it is not optimal, than to stay around on the field thinking about what to do. On the other hand, for clever team play, there is a need to, say, calculate the game position. Imagine you want to attack over the right-hand side of the field but the opponent defense is in good position while the left wing is open to advance. Estimating the game situation like this the agent holding the ball should play a pass back to a free midfielder that is in turn in a position to serve a left forward.

It seems that only taking the currently available game data into account, there is not enough information available for such decisions. We therefore think that besides reactivity a good decision module for a soccer agent needs a deliberative component in one form or another. By deliberation we understand the possibility to reason about actions and make plans about future actions to perform. Moreover, it is not enough to think only about one's own actions. Due to soccer being a team play it has to take possible actions of teammates into account. As in the example of the wing change, the ball holding agent should also reason about what would be the best action for the midfielder receiving the back pass to be able to estimate that the midfielder has a good chance to complete the wing change.

The logic-based programming language GOLOG [Levesque *et al.*, 1997] is an approach to reason about action effects combining explicit agent programming as in imperative programming languages with deliberation. Based on the situation calculus [McCarthy, 1963] GOLOG is able to reason about the world evolving from situation to situation. Over the past years many extensions like dealing with concurrency, exogenous and sensing actions, a continuous changing world and probabilistic projections [G. Lakemeyer, 1999; de Giacomo und H.J. Levesque, 1999; Giacomo *et al.*, 2000; Grosskreutz and Lakemeyer, 2000b; Grosskreutz, 2000; Grosskreutz and Lakemeyer, 2000a; 2001] made GOLOG into an expressive agent programming language. It was used as high-level controller in robotic's applications as a museum tour-guide [Burgard *et al.*, 1998], in computer animation [Funge, 1998], and for low-cost robots like the Lego Mindstorms [Levesque and Pagnucco, 2000].

Combining many of the above mentioned features into a single language, we propose ICPGOLOG as an agent programming language suitable for highly-dynamic multiagent domains like robotic soccer. Using a double pass as an example, we demonstrate in the paper how multirobot coordination can be achieved using ICPGOLOG. We also report on experimental results obtained in the SIMULATION league. However, our main goal is to use the language to control and coordinate the actions of real robots. To this end we have recently integrated ICPGOLOG into our MID-SIZE league robots. We discuss the software architecture and some of the differences in agent modeling in simulated versus physical robots. At the time of the workshop we hope to report on our first experiences using ICPGOLOG on real robots.

The rest of the paper is organized as follows. After briefly introducing relevant aspects of agent control languages in Section 2, we present our system architecture in Section 3 as a hybrid approach combining both the reactive and the deliberative paradigm. As the deliberative part of this architecture, ICPGOLOG is presented in Section 4. We present experimental results in Section 5. Section 6 gives a brief overview of our MID-SIZE software architecture with ICPGOLOG as the high-level controller. We further discuss some of the differences between the SIMULATION and MID-SIZE league regarding high-level control. In Section 7, we give a brief summary and an outlook on future work.

2 Related Work

Below we give a short summary of several architecture approaches applied in ROBOCUP. The classification is based on [Dorer, 1999b] and [Wooldridge, 1999].

Reactive architectures are based on an immediate assignment between perception and action without an explicit description of how a goal can be achieved [Maes, 1990]. Examples are the Subsumption-Architecture [Brooks, 1986], the Situated-Agents [Agre and Chapman, 1990], the Dual-Dynamics approach by Jäger and Christaller [Jaeger and Christaller, 1998], or UML-Statecharts [Arai and Stolzenburg, 2002]. *BDI architectures* are based on the work of Bratman on practical reasoning [Bratman, 1987]. Following Bratman the internal state of an agent is determined by its knowledge about the environment (beliefs), the action facilities the agent is able to choose from (desires) and the current goals (intentions). Representatives for this approach are PRS by Georgeff and Lansky [Georgeff and Lansky, 1987] and the recent Double-Pass Architecture based on mental models [Burkhard, 2001]. *Logic-based architectures* try to obtain a goal-directed plan (sequence of actions) by using a symbolic description and a theorem prover. This manipulation of symbolic data is also called deliberation. One of the most influencing approaches is McCarthy's Situation Calculus [McCarthy, 1963]. Because deliberation is a complex, time consuming process, an optimal plan is only obtained for the situation where planning started, but not necessarily for the current situation. Another system based on Plan Description Languages is introduced in [Iocchi *et al.*, 2000]. Systems following the *hybrid approach* try to combine the advantages of reactive and goal-directed aspects of other architectures. Layers found in most of these models are a reactive layer, a deliberative layer and a modeling layer. Due to using separate modules a coherence problem arises, i.e. there has to be a module making them work together reasonably. Examples are Touringmachines [Ferguson, 1994] or InterRAP [Müller, 1996]. Our own DR-Architecture [Dylla *et al.*, 2002] belongs to this class. *Integrated architectures* aim not only for the combination but the integration of reactive and goal directed behavior. The Situated Automata [Kaelbling and Rosenschein, 1990] and Enhanced Behavior Networks proposed by Dorer [Dorer, 1999a] are examples of this approach.

3 The DR-Architecture

While deliberation has many advantages for decision making of an agent, it has the disadvantage of being slow compared to generating actions in a reactive fashion. In [Dylla *et al.*, 2002] we proposed a hybrid architecture which allows the combination of deliberation with reactivity. In this Section we give an overview of our architecture. This architecture is not only the basis for our SIMULATION team but for the MID-SIZE team as well.

Figure 1 shows the DR-Architecture. From the sensory input we build our world model (gray box in Figure 1). It contains data like the positions of teammates, opponents, or the ball (*Basic World Model*), classification data about the current situation, e.g. good possible pass partners (*Situation Classification*) and group resp. team-level information, e.g. the basic formation of the team (*Group- and Team-level tactics*).

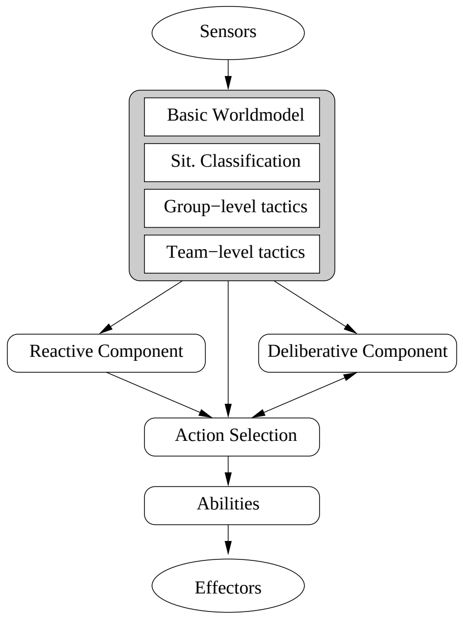


Figure 1: DR-Architecture

The decision module in the DR-Architecture is divided into three modules. To be able to settle an action immediately the *Reactive Component* computes the next action to be executed based on the current game situation. The *Deliberative Component* calculates a plan projecting into the future choosing among the possible action sequences. Section 4 covers this in detail. Having an immediate action and a plan concurring for executing one needs to decide which of both to use. A simple strategy for the *Action Selection* is to use the plan whenever there exists one or execute the action provided by the Reactive Component otherwise. More complex approaches are currently being tested in [Riedel, 2003].

Abilities are the basic actions the agent can perform in the world. In the SIMULATION league the system of UvA-TRILEARN provides actions on three layers of granularity. The fine grained layer contains actions like *kick* or *dash*, which are given by the SOCCERSERVER while the coarse model provides high-level abilities like *dribble* or *intercept ball*. For a detailed description we refer to [de Boer and Kok, 2002]. We use the UvA-TRILEARN system as our basic system.

4 From GOLOG to ICPGOLOG

GOLOG [Levesque *et al.*, 1997] is a high-level programming language for specifying complex tasks like those typically found in robotic scenarios. Similar to ordinary imperative languages, GOLOG offers constructs like *sequence*, *if-then-else*, *while*, and *recursive procedures*. In addition, a *nondeterministic choice* operator allows the robot to choose at runtime from the given alternatives. Another important difference compared to most other programming languages is the notion of a *test condition*, which, in general, can be an arbitrary first-order sentence.

We will not go into any technical details of the semantics of GOLOG except to note that it is based on the *situation calculus* [Reiter, 2001]. From a user's point of view, the following needs to be specified: a set of so-called *fluents*, which are predicates and functions that may change over time like the position of a robot, a set of *primitive actions* like moving to a certain location, *preconditions* and *effects* of primitive actions, and a (first-order) description of the initial state or situation. The user can then write programs which use the given primitive actions and where test conditions refer to fluents. We will see example programs below. Perhaps the most interesting aspect of GOLOG is the ability to project (or simulate) the outcome of a program before actually executing it. This way an agent can evaluate different alternatives and choose the best one for execution.[1]

The features offered by the original GOLOG were soon found to be insufficient for realistic robot application. The first major extension was CONGOLOG [Giacomo *et al.*, 2000], which added the ability to execute actions concurrently and the notion of an *exogenous action*, which is useful to model events outside the robot controller like a low-level routine reporting the current position of the robot. INDIGOLOG [de Giacomo und H.J. Levesque, 1999] extends CONGOLOG by allowing the interleaving of on-line execution and projection.

For robotic domains, however, at least two things were still missing, actions that describe continuous change to, say, reason about the movement of a mobile robot, and noisy sensors and effectors. These features were added in CCGOLOG [Grosskreutz and Lakemeyer, 2000b; 2001] and PGOLOG [Grosskreutz, 2000; Grosskreutz and Lakemeyer, 2000a], respectively. Among other things, these offer constructs that allow an agent to wait for a certain event to occur (*waitFor*), to execute a sub-program which can be blocked at any time once a given condition is no longer satisfied (*withCtrl*), and to execute an action with some probability p and an alternative with probability $1 - p$.

In our current work, we have integrated all these features into a single framework called ICPGOLOG, whose Prolog-implementation is based on an existing implementation of INDIGOLOG. Before turning to applying ICPGOLOG to robotic soccer, here is a brief summary of the language features and the notation used:

- Sequence: a_1, a_2
- Nondeterministic Choice: $a_1; a_2$
- Test: $?(c)$
- Event-Interrupt: $waitFor(c)$
- If-then-else: $if(c, a_1, a_2)$
- While-loops: $while(c, a_1)$
- Condition-bounded execution: $withCtrl(c, a_1)$
- Concurrent actions: $pconc(a_1, a_2)$
- Probabilistic actions: $prob(val_{prob}, a_1, a_2)$
- Probabilistic (off-line) projection: $pproj(c, a_1)$
- Procedures: $proc(name(parameters), body)$

[1]We remark that the language provides a solution to the frame problem [Reiter, 2001].

4.1 A soccer agent in ICPGOLOG

For specifying an agent in ICPGOLOG one starts with a description of the domain knowledge. One has to specify the actions the agent can perform. ICPGOLOG distinguishes between *primitive*, *sensing*, and *exogenous* actions. The primitive actions are those the agent can perform in the world, sensing actions are used to get information about the world. A fluent is assigned to each sensing action which the agent can test to get information about the world. Exogenous actions are actions which occur in the world and to which the agent can react. For each action one has to specify a precondition to state when the action is possible. To describe the effects of an action, one must provide effect axioms, describing which fluents are changed when performing this particular action. Following, we give an example of such a description for the soccer domain.

1. $prim\_fluent(seeBall)$
2. $prim\_fluent(playmode)$
3. $exog\_action(changePlaymode(PM))$
4. $poss(changePlaymode, true)$
5. $causes\_val(changePlaymode(PM),$
   $playmode, PM, legal(PM))$
6. $prim\_fluent(passPartner)$
7. $prim\_action(search\_next\_Passpartner(Player))$
8. $senses(search\_next\_Passpartner(Own),$
   $passPartner)$
9. $cont\_fluent(ballPosition)$
10. $prim\_action(directPass(Sender, Recipient))$
11. $poss(directPass(X, Y),$
    $and(ballOwner(X), seePlayer(Y)))$
12. $causes\_val(directPass(X, Y), ballOwner,$
    $Y, not(passIntercepted(X, Y)))$
13. $initial\_val(ballPosition, [0, 0])$
14. $initial\_ll(ll\_linearBallMove)$

The relational fluent $seeBall$ tells the agent whether it sees the ball or not. It is a primitive fluent in contrast to continuous fluents which will be explained below.

The fluent of Item 2 denotes the server's playmode. Every time the playmode is changed by the SOCCERSERVER an exogenous action is generated by the basic system. The agent notices a playmode change by testing the playmode fluent. The effect axiom of this action (5) changes the value of the playmode fluent when the exogenous action was generated by the basic system and received by the ICPGOLOG agent. The playmode is only changed if a legal playmode is transmitted to the agent otherwise its value remains unchanged. This is denoted by the last argument of the $causes\_val$ predicate in

(5). An example for a precondition is given in (4) stating that a playmode change can always happen.

To play a pass, an agent has to determine to which teammate it should play the pass. Next, the description for a possible pass partner follows. The fluent $passPartner$ is affected by the sensing action $search\_next\_Passpartner(Player)$.

The predicate $senses$ instructs the interpreter to set the fluent $passPartner$ to the respective value.

Fluent $ballPosition$ holds the position of the ball. It changes continuously every cycle and is therefore stored in a special continuous fluent.

In (10) the primitive action for playing a direct pass from the $Sender$ to a pass $Recipient$ is defined. The action only seems reasonable if the sender is in ball possession and knows where the recipient is (11). The action has the effect that the ballholder changes from sender to the recipient, but only if the ball was not intercepted by an opponent player.

Additionally, one has to define the fluent's initial values, e.g. the ball being in the center of the field before kick-off (12). For continuous fluents one has to specify a so-called low-level model (13). Those models are used in projections to determine the correct fluent value. For the ball position a linear approximation of the ball movement is used.

```
proc(soccer_agent,
  pconc( [ withCtrl(playmode=pm_BeforeKickOff,
                place_on_field(playmode) ),
             withCtrl(playmode=pm_Null, wait(1) ),
             withCtrl(playmode=pm_FreeKick_Left,...),
             ...
             withCtrl(playmode=pm_PlayOn,
                play_soccer) ],
  [ execute(next_action), fail].
```

Figure 2: Top-Level structure of the soccer agent

The next step is to provide ICPGOLOG programs to describe agent's behavior. The playmode is the most important parameter for determining a next action. If the mode changes the agent has to react immediately. Within the top level procedure $soccer\_agent$ (see Figure 2) this is modeled by the $withCtrl$ statement. $withCtrl(\mathrm{playmode} = \mathrm{pm\_BeforeKickOff}, \ldots)$ means that the agent should take its position on the field before the game is started. As long as this conditions holds, the agent will call the $place\_on\_field$ procedure. At that moment a condition fails, the respective procedure is interrupted. To be able to catch all possible playmodes we use the $pconc$ statement. By this, the different $withCtrl$ statements are executed concurrently.

```
proc(play_soccer,
  [ try_goal_shot(Own);
    ...;
    try_double_passes(Own);
    try_direct_passes(Own);
    ...        ]).
```

Figure 3: Selecting a behavior

In the case of $playmode = pm\_Play\_On$ the procedure $play\_soccer$ is called (see Figure 3). This procedure encodes the behavior of the agent in normal play. If a plan is applicable with respect to the low-level models in the situation the projection is based on, it will be selected for execution. So far, the order actions are evaluated is fixed. The first successful complex action in this order will be selected, the others below will be ignored. Methods changing this order in a reasonable way are currently under investigation.

```
proc(try_double_passes(Own),
  [ initializePassPartner,
    search_next_Passpartner(Own),
    while ( not(passPartner=nil),
      if ( pproj(has_ball(Own),
                 try_double_pass(Own, passPartner)) >= 0.9,
        [ print("PLANNED DOUBLE PASS."),
          set_next_action(double_pass(Own, passPartner)),    ],
      search_next_Passpartner(Own))
    )]).
```

Figure 4: Module for evaluating several double pass opportunities

In Figure 4 we display the procedure for evaluating double pass possibilities with several pass partners. The agent projects a double pass with a specific partner by $pproj$ and checks whether the projected probability is higher than 90%. In this case the double pass with the computed pass partner is selected and executed otherwise a next pass partner is tried. To coordinate both agents, the pass partner uses the same plan descriptions and projects from the ballholder's view.

```
proc(try_double_pass(Own, TargetPlayer),
  [ look_for_free_space(Own, TargetPlayer),
    directPass(Own, TargetPlayer, pass_NORMAL),
    pconc( [ pconc( receivePass(TargetPlayer),
          intercept_direct_pass(closestOppToPass(TargetPlayer),
                TargetPlayer)),
             if( isBallKickable(TargetPlayer),
               [ kickTo(TargetPlayer, freePos, 0.8),
                 intercept_direct_pass(closestOppToPass(Own),
                   Own)])],
           [ moveToPos(Own, freePos),
             receivePass(Own)])]).
```

Figure 5: Module for projecting one double pass taking opponents into account.

The description of a double pass with a specific teammate is shown in Figure 5. In a double pass situation an opponent is staying right before our agent. To determine a good position where to receive the pass back from the teammate the agents look for a free space behind the opponent. The action $look\_for\_free\_space$ does right this setting the fluent $freePos$ to a particular position in the free region. After that the agent passes the ball to its partner it starts to run towards the free position. Meanwhile, the ball is moving towards his teammate (modeled by $pconc$). It is expected that the closest opponent tries to intercept the moving ball ($intercept\_direct\_pass$). When the second player receives the ball, it kicks it back to the position where the initiator waits for the ball. The opponent's attempt to intercept the ball is modeled for this pass as well.

```
proc(execute_double_pass(Own, TargetPlayer),
  [ look_for_free_space(Own, TargetPlayer),
    directPass(Own, passPartner, pass_NORMAL),
    receivePass(passPartner),
    kickTo(passPartner, freePos, 0.8),
    moveToPos(Own, freePos),
    receivePass(Own),
    setPassFinished,
    setTrySucceeded(true) ].
```

Figure 6: Module for executing a double pass

The execution of the projected path is shown in Figure 6. The ballholder has to look again where his teammate and the free space is, because the world might have changed. The actions with argument Own are executed by the agent itself while the actions with $passPartner$ as argument are supposed to be executed by the respective teammate.

To illustrate two more important features we show in Figure 7 two procedures for projecting a direct pass with two possibilities to model opponent behavior. By the $waitFor$ construct in Figure 7, the agent waits until it is able to stop the ball or the ball is too far away, probably it was intercepted and the current action is not reasonable anymore. With this construct we can test the end of the pass action. Another concept is the $prob$ statement. We can test two different models of an opponent intercepting our pass. With probability 0.7 the pass may be intercepted using model $intercept\_direct\_pass1$ and with probability 0.3 using $intercept\_direct\_pass2$. In the first alternative the model can be very pessimistic, i.e. the opponent will perform a real sophisticated intercept action, or optimistic in the other case.

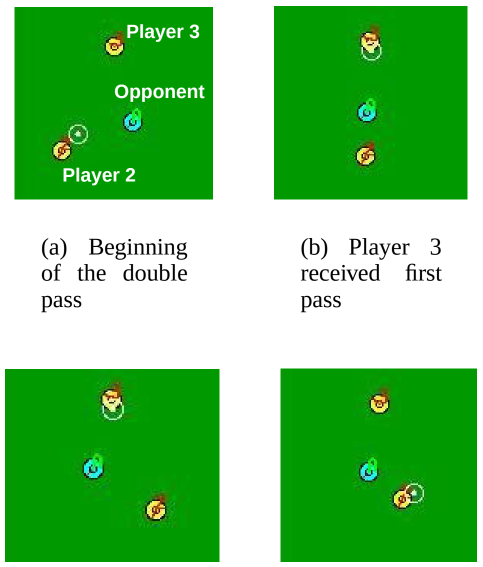
4.2 Executing a Double Pass

To illustrate how ICPGOLOG works in practice we give an example execution of a double pass. We refer to the procedures given above. We first introduce the example setting and show how a multiagent plan is generated and executed in ICPGOLOG by an execution trace of the program *try_double_passes*.

Our scenario is the following. Player 2 (the lower yellow player in Figure 8) wants to outplay the opponent with a double pass. Player 3 (upper yellow player in Figure 8) is in a good position to play a double pass with Player 2. Player 2

```
proc(try_direct_pass(Own, TargetPlayer),
  [ directPass(Own, TargetPlayer, pass_NORMAL),
    pconc(prob_intercept_direct_pass(closestOpp-
         ToPass(TargetPlayer), TargetPlayer),
    waitFor(or(ball_near_player(TargetPlayer),
                  ball_far(TargetPlayer))))]).
proc(prob_intercept_direct_pass(Opp, TargetPlayer),
    prob(0.7, intercept_direct_pass1(Opp, TargetPlayer),
    intercept_direct_pass2(Opp, TargetPlayer))).
```

Figure 7: Module for projecting a direct pass with probabilistic opponent behavior



(a) Beginning of the double pass

(b) Player 3 received first pass

(c) Player 2 moves to receive position

(d) Player 2 receives the second pass

Figure 8: Double pass scenario

therefore initiates the double pass by playing a direct pass to Player 3. Thereafter, Player 2 has to run to the position where it can receive the pass from Player 3 (Figure 8(b)). Player 3 receives the ball and should pass it back to Player 2 if Player 2 itself is near the reception position (Figure 8(c)). Finally, Player 2 receives the ball (Figure 8(d)).

To make it more concrete, we show the ICPGOLOG execution trace of the described scenario in Figure 9. Although we are able to reason about the behaviors of opponents by appropriate models as well, we leave out this detail here. The left column of this figure shows the trace for Player 2 which initiates the pass, the right one for Player 3. Player 2 starts by getting a new world model and intercepting the ball to be able to play the first pass (lines 1 – 6). After the successful intercept action both agents start the procedure $try\_double\_passes(Own)$ (refer to Figure 4). The variable Own is set to Player 2 for both agents. Player 3 therefore plans all actions of the double pass from Player 2's point of view. Of course, in the execution each agent performs only actions regarding itself.

The first action in this procedure is to find a pass partner. After resetting the fluent $passPartner$, the agents projects all possible partners for playing a pass. This is expressed by the $pproj$ statement in Figure 7. This corresponds to the lines 7 to 13 in Figure 9. If this projection is successful with probability 0.9 the procedure $execute\_double\_pass$ is called. As one can observe in lines 14 and 15 in Figure 9, both players are executing the action `directPass`. The execution system can determine by the command `nextSkill(2)` that this action is for Player 2. Player 3 will not perform the action in the real world.

```
                Player 2                                     Player 3


 1 send(getBasicWorldModel, true)
   send(nextSkill(2), intercept)
   WAITING FOR EXOGENOUS ACTIONS...
   setPlayerProj(2,[-28.34,-2.33],...)
 5 waitedIntercept
   send(getBasicWorldModel, true)             send(getBasicWorldModel, true)
   initializePassPartner                      initializePassPartner
   setPassPartner(3)                          setPassPartner(3)
   Prob. Proj. Test                           Prob. Proj. Test
10  (# of initial configs: 1,                   (# of initial configs: 1,
    unsorted/sorted # of traces:1/1).           unsorted/sorted # of traces:1/1).
   Prob. Proj. Test (cached result).
   write(PLANNED DOUBLE PASS.)                write(PLANNED DOUBLE PASS.)
   send(nextSkill(2),                         send(nextSkill(2),
15   [directPass, [3, pass_NORMAL]])            [directPass, [3, pass_NORMAL]])
   setBallProj([-27.50,-3.88],...)            setBallProj([-28.50, -3.00], ...)
   send(nextSkill(3), receivePass)            send(nextSkill(3), receivePass)
                                              WAITING FOR EXOGENOUS ACTIONS...
   setBallProj([-23.27,-11.44],...)           setBallProj([-23.19, -11.43],...)
20 send(nextSkill(3),                         send(nextSkill(3),
     [kickTo, [[-18.71,-1.88],0.4])             [kickTo, [[-21.42, 0.98],0.4])
                                              WAITING FOR EXOGENOUS ACTIONS...
   setBallProj([-27.50,-3.88],...)            setBallProj([-23.26, -8.75], ...)
   send(nextSkill(2),                         send(nextSkill(2),
25   [moveToPos,[[-18.71,-1.88],...]])          [moveToPos,[[-21.42,0.98],...]])
   WAITING FOR EXOGENOUS ACTIONS...
   setPlayerProj(2,[-18.71,-1.88],...)        setPlayerProj(2,[-21.42,0.98],...)
   send(nextSkill(2), receivePass)            send(nextSkill(2), receivePass)
   WAITING FOR EXOGENOUS ACTIONS...
30 setBallProj([-20.57,3.19],...)             setBallProj([-3.37, 2.97],...)
   send(getBasicWorldModel, true)             send(getBasicWorldModel, true)
   send(nextSkill(2), intercept)              send(nextSkill(2), intercept)
   setPlayerProj(2,[-20.96,3.47],...)         setPlayerProj(2,[-23.30,-6.25],...)
   waitedIntercept                            waitedIntercept
35 setPassFinished                            setPassFinished
   setTrySucceeded(true)                      setTrySucceeded(true)
   send(nextSkill(2), intercept)
   WAITING FOR EXOGENOUS ACTIONS...
   setPlayerProj(2,[-21.01,3.62],...)
40 waitedIntercept
```

Figure 9: Execution traces of the pass sender and receiver in the double pass situation

We now enter phase 2 of our double pass (Figure 8(b)) where the first pass is to be received by Player 3. Again, both player settle the same action (`receivePass`). To synchronize actions of both players the execution system waits until some condition meets denoting the end of the respective action. In our example the reception of the first pass is acknowledged by an exogenous event "*received pass*".

This is modeled by an exogenous action (line 18 in Figure 9, `WAITING FOR EXOGENOUS ACTION`). The pass back from Player 3 to Player 2 is not modeled by a direct pass. Instead, a $kickTo$ action is performed to a position calculated by $look\_for\_free\_space$ in Figure 6, i.e. a position in a free region behind the opponent. Note that the goal position of the $kickTo$ command slightly differs in both traces. This can be explained by the different world models of the resp. agent based on which this calculation is done.

Figure 8(c) shows the situation when Player 2 is near the calculated receive position. Finally, Player 2 receives the pass (Figure 8(d)). As stated above, there can be small differences in values derived from agent's world model. Therefore, to ensure that Player 2 receives the ball, it performs an intercept action in the end.

In this example we show a multiagent plan for a double pass. This plan does not use explicit communication to coordinate the agents involved. The execution of this plan is possible because both player reason about the same actions. Player 3 in the example generates the plan from the ballholder's point of view and comes to the same conclusion as Player 2. So, Player 3 identifies itself to be the best pass partner for Player 2. Multiagent coordination like this only works if agents' world models are similar and not too uncertain.

For our implementation we used the PROLOG system ECLIPSE [ECLiPSe, 2002] for the ICPGOLOG interpreter. The primitive ICPGOLOG actions we used in the double pass example are sent to the basic agent that is connected to the SOCCERSERVER. Our basic agent is an extension of the UvA-TRILEARN system from the University of Amsterdam [de Boer and Kok, 2002]. The PROLOG system communicates with the basic agent via shared memory. The high-level agent receives its world model from the basic agent and sends the next action to be executed to it which in turn translates it to SOCCERSERVER commands.

Finally, we remark that for our implementation of ICPGOLOG, we needed to solve a problem that is common to most GOLOG variants, but which becomes quite critical in real-time environments like robotic soccer. The issue is that in order to evaluate a test condition, the GOLOG interpreter usually applies what is called *regression*, which means, roughly, that the interpreter needs to transform the test condition to one that can be evaluated in the initial situation, taking into account *all* the actions which have happened so far. In our case the number of actions to be considered in this process quickly grows into the thousands, which leads to severe computational problems. As an alternative, Lin and Reiter [Lin and Reiter, 1997] proposed what they call *progression*, which means that after an action has been executed, the description of the initial situation is updated to reflect the effects of the action. This way tests can always be evaluated directly against the current state of the world. Unfortunately, Lin and Reiter's proposal does not lend itself to an efficient implementation because the size of the world description easily grows exponentially when applying progression. For our domain we developed a simpler, set-based form of progression which is more restricted than Lin and Reiter's version but which is computationally viable. In fact we were able to gain an exponential speedup compared to using regression. For details we refer the reader to [Dylla *et al.*, 2003].

5 Experimental Results

We tested the above described framework in the scenario of ROBOCUP SIMULATION league. Our agent programming language is expressive at the cost of higher runtimes. We therefore tested on which level of abstraction deliberation is reasonable. Three models differing in their level of granularity were implemented. The classification is roughly adopted from the UvA-TRILEARN system. Fine granular actions are the basic actions provided by the SOCCERSERVER, e.g. $kick$, $dash$ or $turn$ valid for one cycle each. The medium granular model contains actions like $dashToPoint$ and the coarse consists of action on the level $interceptBall$ or $directPass$ continuing for several server cycles. In all tests the task was to project and execute an action. The results are based on tests where only few agents connected to the SOCCERSERVER, not two whole teams.

The $GoalShot$ tests a shot in each corner of the goal, while concurrently simulating the behavior of the goalie and the closest opponent to the ball's trajectory. The $DirectPass$ procedure models a direct pass to two different teammates with an opponent trying to reach the pass-way. The test of more pass options leads to a linear runtime increase per additional teammate. Within $Doublepass$ two different possibilities of playing a double pass are tested with the opponent closest to the pass-way trying to intercept the ball. For coordination purposes the teammate calculates his behavior by putting himself in the ballholder's place. At last the $DirectPassWithPO$ models the same as $DirectPass$ with the difference the opponent having four probabilistic choices of how to behave. Some test results are shown in Figure 10. For a complete overview we refer to [Jansen, 2002; Dylla *et al.*, 2003].

The runtime results in Figure 10 suggest that the use of ICPGOLOG's projection mechanism is currently only feasible computationally at a high level of abstraction like in the coarse model. Here, the use of continuous fluents for projecting the ball position, for example, brings runtime advantages over the fine grained model, where those continuous fluents could not be used.

6 From Simulations towards Real Robots

While the SIMULATION league certainly provides a rich environment to test ideas in multiagent coordination, our main goal is to apply them to real robots. For that purpose we recently acquired a team of robots for the MID-SIZE league. Given the experience of other teams that "off-the-shelf" robots must be completely reengineered for ROBOCUP's MID-SIZE to be competitive, we decided to develop our own robotic platform from scratch [Wunderlich and

GoalShot	fine	middle	coarse
average no of actions / plan	101	80	38
time [s]	1.91	1.12	0.31

DirectPass	fine	middle	coarse
average no of actions / plan	93	40	39
time [s]	1.8	0.48	0.25

Doublepass	fine	middle	coarse
average no of actions / plan	319	319	86
time [s]	6.16	5.8	0.69

DirectPassWithPO	fine	middle	coarse
average no of actions / plan	95	42	41
time [s]	7.22	2.95	0.67

Figure 10: Runtime results in the SIMULATION-League

Dylla, 2002]. The intention was to develop robots competitive in ROBOCUP which can also be used in office domains for service-robot applications. The platform has a size of 39 cm $\times$ 39 cm $\times$ 40 cm (Figure 11). For power supply we have two 12 V lead-gel accumulators with 15 Ah each onboard. The battery power lasts for approximately one hour at full charge. The robot has a differential drive, the motors have a total power of 2.4 kW. This power provides us with a top speed of 3 m/s and 1000°/s by a total weight of approximately 50 kg.

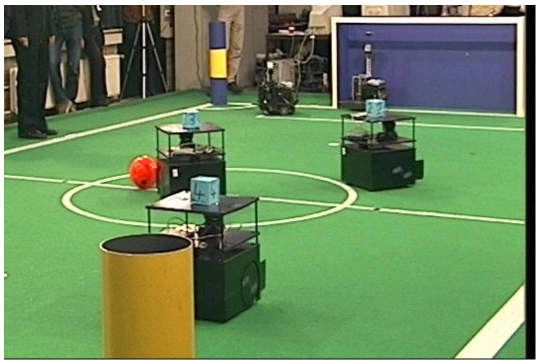


Figure 11: Robots of the AllemaniACs MID-SIZE team

Onboard we have two Pentium III PC's at 500 MHz running Linux, one equipped with a framegrabber for a Sony EVI-D100P camera mounted on a pan/tilt unit. Our other sensor is a 360° laser range finder with a resolution of 0.75 degree at a frequency of 20 Hz. For communication a WLAN adapter based on IEEE 802.11b is installed.

Figure 12 gives a schematic overview of the different software components and the data and command flow between them. The modules *motor*, *kicker* and the *camera's pan/tilt* are drivers for the actuators, *laser* and *camera* are driver modules for the sensors. The collision avoidance module *colli* uses the A^* algorithm to compute a collision free path to a

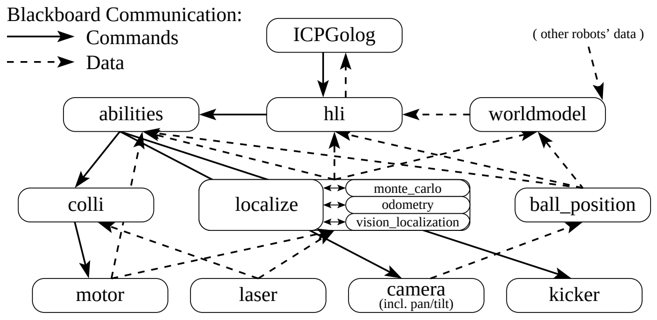


Figure 12: Components of our software system

target point every 50 ms. Localization is based on the fusion of several methods. The module is dominated by a Monte Carlo approach using the laser data. Due to complexity of the Monte Carlo approach update frequencies are lower for the localization. It turned out that a frequency of 2 up to 4 is sufficient for good localization. Between position updates of *localize* the robot uses it's odometry for position calculations. Additionally, information extracted from a vision algorithm using probability maps [Jones and Rehg, 1999] is used for resolving field symmetries. The ball is perceived by the *ball_position* module. The *ability* module defines different basic actions the robot can perform. Abilities such as *goto* and *dribble* send respective commands to the collision avoidance module which in turn actuates the motor, camera and kicker.

Our world model basically holds (quantitative) information about the positions of the robots and the ball. Moreover, information about the game state like the robot's role in the play or the current play mode is represented as well. Our interface to the high-level control is encoded in the module *hli* (high-level interface) This interface is a wrapper program between the PROLOG system of our high-level control and the basic robot system which is implemented in C++. For interprocess communication we use a blackboard communicating via shared memory. For inter-robot communication UDP is used.

To ensure safe communication via UDP the blackboard system has its own security layer. This feature offers the possibility to implement a global world model. Figure 13 shows the communication infrastructure of inter-robot data exchange. An external data synchronization module (*sync*) receives local world information from each robot and transmits them to a global blackboard to which the global world model is connected. Each robot in turn receives the global world model data in the opposite direction. To control the local software modules, *rccc* (*r*obo*c*up *c*ontrol *c*enter) was implemented. It offers the possibility to start the software on each robot by a different communication channel using remote procedure calls.

The possibilities to model robot behavior are much more restricted compared to SIMULATION league. One obvious reason is that the world model the robot can rely on is much more uncertain than in SIMULATION league, e.g. regarding one's own position or the ball position. Moreover, it is harder to get informations about, say, opponent positions. Another

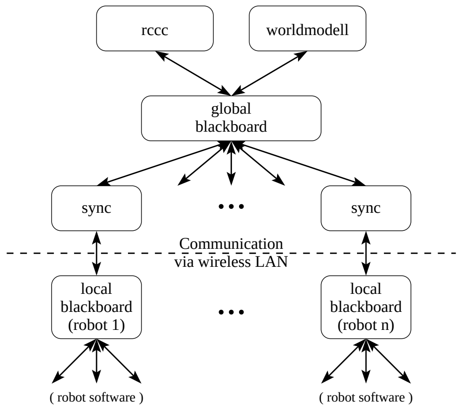

rccc
worldmodell
global blackboard
sync
sync
Communication via wireless LAN
local blackboard (robot 1)
local blackboard (robot n)
(robot software)
(robot software)

Figure 13: Global connectivity of the robots' local software systems

problem is related to the actuators. While in SIMULATION league it is rather simple to kick the ball or to play a pass it is much more complicated in MID-SIZE league. On the other hand, the coordination of the team should be easier than in the SIMULATION league since only four robots have to be coordinated. For a pass only two teammates have to be considered as possible pass receivers whereas in SIMULATION this number is normally much higher. The robots, in general, have more time to "think" about what action to perform because there is not a strict decision cycle as in SIMULATION league. Thus, the calculation times shown in Figure 10 seem not to be too problematic. Another difference between the SIMULATION and MID-SIZE league concerns communication. In the MID-SIZE league no restriction of what amount of data may be communicated between the robots via wireless LAN exists whereas it is limited to a few bytes in the SIMULATION league. Therefore communication may be used for more reliable coordination of the robots.

7 Discussion

We introduced the framework of ICPGOLOG for developing deliberative agents. By integrating features like concurrency, exogenous actions, continuous change and the possibility to project into the future we are able to model agents for highly-dynamic environments like robotic soccer. We showed how multiagent coordination can be achieved without communication. The requirement for coordination without communication is that the world models of the agents do not differ too much. In the tested cases of the SIMULATION league the agents' world models are not too uncertain due to the good sensory information given by the SOCCERSERVER. Therefore, this kind of multiagent coordination works well. In the MID-SIZE league, where world models are not that elaborate and more uncertain due to sensor noise, this approach to multiagent coordination might be problematic. It is likely that communication between robots will play a much bigger role here. It remains to be seen how much communication is necessary for coordinated actions of real robots. However, it does not necessarily only get harder when moving from simulations to real robots. For example, in the MID-SIZE league the real-time requirement for deciding on the next action is not as strict as in the SIMULATION league. Therefore, the robot has more time for generating plans with ICPGOLOG. Moreover, fewer agents have to be coordinated. We are currently evaluating and testing what works best for our MID-SIZE robots and hope to report on our experiences at the time of the workshop. Ultimately we also want to get deeper understanding of the similarities and differences between the SIMULATION and MID-SIZE leagues.

We end the paper with remarks on two other research issues currently under investigation. The first concerns the action selection mechanism. Having two decision components (reactive and deliberative) competing for the next action to be executed we have to select which action to use. This affects also the problem of the validity of ICPGOLOG plans over time. Once a plan is generated, it will be executed for a number of cycles. The action selection also has to check if the world evolved as anticipated in the plan. In [Riedel, 2003] those questions are investigated. Again it is likely that there will be significant differences between simulated and real soccer agents.

The second is about how to select optimal actions. In this regard Boutilier et al. [Boutilier *et al.*, 2000a] recently proposed a decision-theoretic GOLOG dialect based on Markov Decision Processes. We plan to integrate decision-theoretic concepts into our ICPGOLOG framework as well. To this end we are currently working on speeding up the computation of optimal policies in decision theoretic GOLOG [Ferrein *et al.*, 2003] by incorporating the notion of macro actions considered in the MDP literature [Boutilier *et al.*, 2000b; Hauskrecht *et al.*, 1998; Sutton *et al.*, 1999].

References

[Agre and Chapman, 1990] P. E. Agre and D. Chapman. What are plans for? In P. Maes, editor, *Designing Autonomous Agents*, pages 17–34. The MIT Press, San Francisco, CA, 1990.

[Arai and Stolzenburg, 2002] Toshiaki Arai and Frieder Stolzenburg. Multiagent systems specification by UML statecharts aiming at intelligent manufacturing. In *Proceedings of the 1st International Joint Conference on Autonomous Agents & Multi-Agent Systems*, pages 11–18, Bologna, Italy, 2002. ACM Press. Volume 1.

[Boutilier *et al.*, 2000a] C. Boutilier, R. Reiter, M. Soutchanski, and S. Thrun. Decision-theoretic, high-level agent programming in the situation calculus. In *AAAI'2000*, 2000.

[Boutilier *et al.*, 2000b] C. Boutilier, R. Reiter, M. Soutchanski, and S. Thrun. Decision-theoretic, high-level agent programming in the situation calculus. In *Proc. AAAI-2000*, 2000.

[Bratman, 1987] M. E. Bratman. *Intentions, Plans, and Practical Reason*. Harvard University Press, 1987.

[Brooks, 1986] R. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, pages 14–23, April 1986.

[Burgard *et al.*, 1998] W. Burgard, A.B. Cremers, D. Fox, D. Hähnel, G. Lakemeyer, D. Schulz, W. Steiner, and S. Thrun. The interactive museum tour-guide robot. In *Proceedings of the AAAI 15th National Conference on Artificial Intelligence*, 1998.

[Burkhard, 2001] Hans-Dieter Burkhard. Mental models for robot control. Dagstuhl Workshop on Plan-based Control of Robotic Agents, 2001.

[de Boer and Kok, 2002] Remco de Boer and Jelle Kok. The Incremental Development of a Synthetic Multi-Agent System: The UvA Trilearn 2001 Robotic Soccer Simulation Team. Master's thesis, University of Amsterdam, Februar 2002.

[de Giacomo und H.J. Levesque, 1999] G. de Giacomo und H.J. Levesque. An incremental interpreter for high-level programs with sensing. In F. Pirri H. Levesque, editor, *Logical Foundations for Cognitive Agents*, pages 86–102. Springer, 1999.

[Dorer, 1999a] Klaus Dorer. Behavior networks for continuous domains using situation-dependent motivations. In Dean Thomas, editor, *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI-99-Vol2)*, pages 1233–1238, S.F., July 31–August 6 1999. Morgan Kaufmann Publishers.

[Dorer, 1999b] Klaus Dorer. *Motivation, Handlungskontrolle und Zielmanagement in autonomen Agenten*. PhD thesis, Albert-Ludwigs-Universität Freiburg, Freiburg, December 1999.

[Dylla *et al.*, 2002] F. Dylla, A. Ferrein, and G. Lakemeyer. Acting and Deliberating using Golog in Robotic Soccer – A Hybrid Approach. In *Proc. 3rd International Cognitive Robotics Workshop (CogRob 2002)*. AAAI Press, 2002.

[Dylla *et al.*, 2003] F. Dylla, A. Ferrein, N. Jansen, and G. Lakemeyer. Progression in the Framework of GOLOG. KBSG, Aachen University, in preparation, 2003.

[ECLiPSe, 2002] ECLiPSe. (Version 5.5) - The ECRC Constraint Logic Parallel System. http://www.icparc.ic.ac.uk/eclipse, 2002.

[Ferguson, 1994] I. A. Ferguson. Integrated control and coordinated behavior: A case for agent models. In M. J. Wooldridge and N. R. Jennings, editors, *Intelligent Agents: ECAI-94 Workshop on Agent Theories, Architectures, and Languages*, pages 203–218. Springer, Berlin,, 1994.

[Ferrein *et al.*, 2003] A. Ferrein, C. Fritz, and G. Lakemeyer. Extending DTGolog with options. In *Proc. IJCAI-03*, 2003. to appear.

[Funge, 1998] J. Funge. *Making Them Behave: Cognitive Models for Computer Animation*. PhD thesis, University of Toronto, Toronto, Canada, 1998.

[G. Lakemeyer, 1999] G. Lakemeyer. On sensing and off-line interpreting in golog. In H. Levesque und F. Pirri, editor, *Logical Foundations for Cognitive Agents*. Springer, 1999.

[Georgeff and Lansky, 1987] M. P. Georgeff and A. L. Lansky. Reactive reasoning and planning. In *The Proceedings of AAAI-87*, pages 677–682, Seattle, 1987.

[Giacomo *et al.*, 2000] Giuseppe De Giacomo, Yves Lésperance, and Hector J. Levesque. ConGolog, A concurrent programming language based on situation calculus. *Artificial Intelligence*, 121(1–2):109–169, 2000.

[Grosskreutz and Lakemeyer, 2000a] H. Grosskreutz and G. Lakemeyer. Turning high-level plans into robot programs in uncertain domains. In *ECAI'2000*, 2000.

[Grosskreutz and Lakemeyer, 2000b] Henrik Grosskreutz and Gerhard Lakemeyer. cc-Golog: Towards more realistic logic-based robot controllers. In *Proceedings of the 7th Conference on Artificial Intelligence (AAAI-00) and of the 12th Conference on Innovative Applications of Artificial Intelligence (IAAI-00)*, pages 476–482, Menlo Park, CA, 2000. AAAI Press.

[Grosskreutz and Lakemeyer, 2001] H. Grosskreutz and G. Lakemeyer. Online-Execution of ccGolog Plans. In *IJCAI'2001*, 2001.

[Grosskreutz, 2000] H. Grosskreutz. Probabilistic projection and belief update in the pGolog framework. In *Second International Cognitive Robotics Workshop*, 2000.

[Hauskrecht *et al.*, 1998] M. Hauskrecht, N. Meuleau, L. Kaelbling, T. Dean, and C. Boutilier. Hierarchical solutions of MDPs using macro-actions. In *Proc. UAI 98*, 1998.

[Iocchi *et al.*, 2000] Luca Iocchi, Daniele Nardi, and Riccardo Rosati. Planning with sensing, concurrency, and exogenous events: Logical framework and implementation. In Anthony G. Cohn, Fausto Giunchiglia, and Bart Selman, editors, *KR2000: Principles of Knowledge Representation and Reasoning*, pages 678–689, San Francisco, 2000. Morgan Kaufmann.

[Jaeger and Christaller, 1998] H. Jaeger and T. Christaller. Dual dynamics: Designing behavior systems for autonomous robots, 1998.

[Jansen, 2002] Norman Jansen. A framework for deliberation in uncertain, highly dynamic environments with real-time requirements. Master Thesis, in German, Knowledge Based Systems Group, Aachen University, Aachen, Germany, 2002.

[Jones and Rehg, 1999] M. J. Jones and J. M. Rehg. Statistical color models with application to skin detection. In *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, volume 1, pages 274–280, 1999.

[Kaelbling and Rosenschein, 1990] L. P. Kaelbling and S. J. Rosenschein. Action and planning in embedded agents. In Pattie Maes, editor, *Designing Autonomous Agents*, pages 35–48. MIT Press, Cambridge (MA), 1990.

[Levesque and Pagnucco, 2000] H. Levesque and M. Pagnucco. Legolog: Inexpensive experiments in cognitive robotics. In *Proc. 2nd International Cognitive Robotics Workshop (CogRob-00)*. ECAI-00, 2000.

[Levesque *et al.*, 1997] Hector J. Levesque, Raymond Reiter, Yves Lesperance, Fangzhen Lin, and Richard B. Scherl. GOLOG: A logic programming language for dynamic domains. *Journal of Logic Programming*, 31(1-3):59–83, 1997.

[Lin and Reiter, 1997] F. Lin and R. Reiter. How to progress a database. *Artificial Intelligence*, 92:131–167, 1997.

[Maes, 1990] Patti Maes. Situated agents can have goals. In Patti Maes, editor, *Designing Autonomous Agents*, pages 49–70. MIT Press, 1990.

[McCarthy, 1963] J. McCarthy. Situations, actions and causal laws. Technical report, Stanford University. Reprinted 1968 in Semantic Information Processing (M.Minsky ed.), MIT Press, 1963.

[Müller, 1996] Jörg P. Müller. The design of intelligent agents. In *Lecture Notes in AI*, volume 1177. Springer, 1996.

[Reiter, 2001] R. Reiter. *Knowledge in Action*. MIT Press, 2001.

[Riedel, 2003] B. Riedel. Developing similarity measures for the comparison of game situations in Robocup. Master Theses, in progress, in German, Knowledge Based Systems Group, Aachen University, Germany, 2003.

[Sutton *et al.*, 1999] R. Sutton, D. Precup, and S. Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Journal of Artificial Intelligence*, 1999.

[Wooldridge, 1999] Michael Wooldridge. Intelligent agents. In Gerhard Weiss, editor, *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*, pages 27–78. The MIT Press, Cambridge, MA, USA, 1999.

[Wunderlich and Dylla, 2002] Jost Wunderlich and Frank Dylla. Technical description of the allemaniacs soccer robots. Technical report, LTI / KBSG, Aachen University, Germany, 2002. in German.

Case-Based Reasoning for Planning and World Modeling in the RoboCup Small Size League

Cynthia Marling, Mark Tomko, Matthew Gillen, David Alexander, and David Chelberg
School of Electrical Engineering and Computer Science
Ohio University, Athens, Ohio 45701, USA
Phone: +1 (740)593-1246, FAX: +1 (740)593-0007
Email: {marling,mark.tomko.1,matthew.gillen.1,david.r.alexander.1, chelberg}@ohio.edu

Keywords: planning, world modeling, RoboCup, case-based reasoning

Abstract

This paper presents three case-based reasoning (CBR) prototypes developed for the RoboCats, a team of five soccer playing robots in the RoboCup small size league. CBR is used to help the RoboCats plan individual moves and team strategies, as well as to model the world of the playing field. More specifically, the case-based reasoners position the goalie, select team formations, and recognize game states for the RoboCats. This paper also discusses issues and opportunities for CBR in RoboCup and other endeavors in which physical agents must operate in dynamic, real-time environments.

1 Introduction

RoboCup is an international, interdisciplinary research project in which physical agents compete in dynamic real-time environments. The game of robotic soccer provides a backdrop and a test bed for research in artificial intelligence, computer vision and robotics [Birk *et al.*, 2002; The RoboCup Federation, 2002]. At present, RoboCup soccer comprises separate leagues for teams of small size robots, middle size robots, four-legged robots, and humanoid robots, plus a simulation soccer league. The ultimate goal of RoboCup is to build a team of humanoid robots capable of beating the best human soccer team in the World Cup by the year 2050. Between now and then, many issues in designing physical agents for dynamic real-time environments will need to be explored and resolved.

Case-based reasoning (CBR) is a paradigm in which solutions to current problems are found by recalling, adapting, and reusing solutions to similar problems encountered in the past [Kolodner, 1993]. A system's (or a person's) experiences are encapsulated as cases that are organized to facilitate their recall whenever they would prove useful again. Once a lesson has been learned, such as how to shoot a penalty kick, or that an opponent with a ball should be blocked, that lesson can be quickly put to use whenever a penalty kick is called or an opponent needs to be blocked. Rather than deliberate about how to overcome all potential problems, a case-based reasoner narrows its focus to problems known to have occurred in its world. One might view this as narrowing the search space to more quickly reach viable solutions and/or as employing a more human-like approach to problem solving.

When Agent Team (AT) Humboldt became *weltmeisters* of the RoboCup simulation league in 1997, they noted that agent learning with CBR was one of the primary research interests of their group [Burkhard *et al.*, 1998]. By 1998, when they returned as vice champions, they had experimented with CBR for both off-line learning, or agent training, and on-line learning, or adaptation during games [Gugenberger *et al.*, 1999]. They used CBR for dynamic situation assessment, comparing a player's current view of the field, as provided by the simulator league's SoccerServer, to game situations that had previously occurred. The most similar past situations were used to determine where the player should move to be most effective at present [Wendler and Lenz, 1998]. Since this early effort, little mention has been made of CBR in this domain.

We are currently integrating CBR into the Ohio University RoboCats, a team that competes in the RoboCup small size league. In this league, a team consists of five robots, each of which may be no larger than 18 centimeters in diameter and 15 centimeters high. We have built preliminary prototypes, in simulation mode, to position the goalie, select team formations, and recognize game states. Based on the insights gained from our experiences with these prototypes, we are now building case-based reasoners for planning team strategies and for opponent modeling. These reasoners will be fully integrated into our robotic soccer team in time for the next Robot Soccer World Cup in Padua, Italy, to be held in July, 2003. This paper describes our completed prototypes, discusses the issues involved, and describes the opportunities we see for CBR in RoboCup. We believe that these issues and opportunities are also relevant for the design of other physical agents that must operate autonomously in dynamic real-time environments.

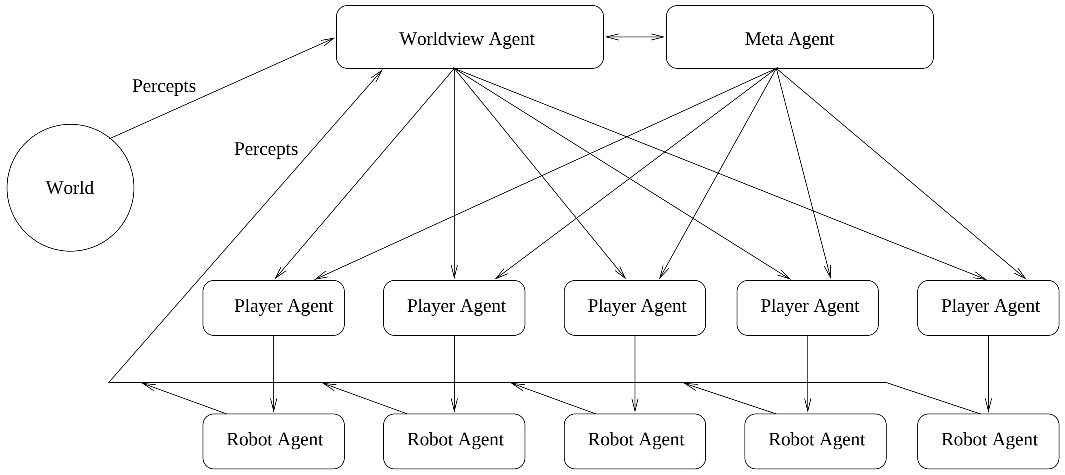


Figure 1: The RoboCats Architecture

2 CBR Prototypes for the RoboCats

2.1 The RoboCats

The RoboCats employ a hybrid hierarchical, schema-based architecture that distributes planning throughout the team, as fully described in [Gillen *et al.*, 2002]. The CBR prototypes, described next, fit within this architecture. An overview of the RoboCats architecture is shown in Figure 1. In the RoboCup small-size league, a camera mounted above the playing field provides vision for the whole team. Raw vision updates from the camera are input to a Worldview Agent. The Worldview Agent analyzes the vision data to determine the state of the game, and makes its analyses available to the Meta Agent and to the five Player Agents. The Meta Agent makes high-level decisions for the team as a whole, much as a human coach would do. It determines team formations and assigns individual roles to each Player Agent. The Meta Agent and the Player Agents are focal points for CBR.

There is one Player Agent and one Robot Agent per physical robot. The Player Agent, given a role, determines how to best fulfill that goal using strategies encoded as schemas. Our view of agency includes, but is not limited to, physical agency in which individual robots act as an agents. The schemas may also be viewed as hierarchically ordered software agents. Higher level schemas generate plans to accomplish higher level goals, like scoring goals or blocking shots. Lower level schemas generate plans to accomplish necessary subgoals, like moving to particular locations. There may be multiple schemas capable of achieving the same goal or subgoal, so selection of schemas involves evaluating both the desirability of the schema's goal and the feasibility of the schema's strategy for accomplishing that goal. Schemas dealing with physical control of the robot hardware are grouped together into the Robot Agents. The most reactive behaviors, like those for trajectory control, are also included in the Robot Agents.

Our first CBR prototype, which positions the goalie, was built as part of a Player Agent. The other two, which select team formations and recognize game states, are included in the Meta Agent. The prototypes were constructed and tested in simulation mode. Because the rules and environment of the simulation league differ from those of the small size league, we built our own custom simulator for the RoboCats. Our simulator models the physical properties of our robots and the ball and calculates the effects of collisions, kicking, and dribbling. It allows us to test experimental software modules more thoroughly than would be possible on our physical robots. Our system architecture makes communications with the simulator transparent. The same agent code runs on either simulator or physical robot, when conditionally compiled to account for differences in command paths.

2.2 Positioning the Goalie

Our first prototype uses past experience to determine where the goalie should move to successfully block an opponent's shot. A case is derived from a snapshot of the goalie's half of the field. From the raw vision data, values are derived for: the position and orientation of the goalie, the position and orientation of each attacker and defender, and the position of the ball. A case contains the initial positions and orientations as the description of the world situation. It also contains the position and orientation the goalie assumed in response to the situation, whether that move resulted in success or failure in blocking the shot, and the position of the ball after the attempted block. An example case, and its associated graphical view, are depicted in Table 1 and Figure 2. In Table 1, positions are represented by an (x,y) coordinate system, in which (0,0) represents the upper left hand corner of the half-field and (137,150) represents the lower right hand corner. Orientations are expressed in degrees of rotation from the *x* axis. In Figure 2, we can see how the goalie moves slightly out and to the left to successfully block a shot, causing the ball to move out in front of the goalie.

Cases are organized in the case base according to the location of the ball on the field. To speed up case retrieval, the case base is partitioned by region within the half-field. There are three regions: near, middle and far. A case that lies on or near regional boundaries is counted as being in both re-

Goalie's Initial Position	9 92
Goalie's Initial Orientation	118
Attacker's Initial Position	73 141
Attacker's Initial Orientation	309
Defender's Initial Position	34 60
Defender's Initial Orientation	103
Ball's Initial Position	65 131
Goalie's New Position	11 89
Goalie's New Orientation	118
Direction of Goalie Move	Left
Ball's New Position	42 87
Outcome	Success

Table 1: An Example Goalie Case

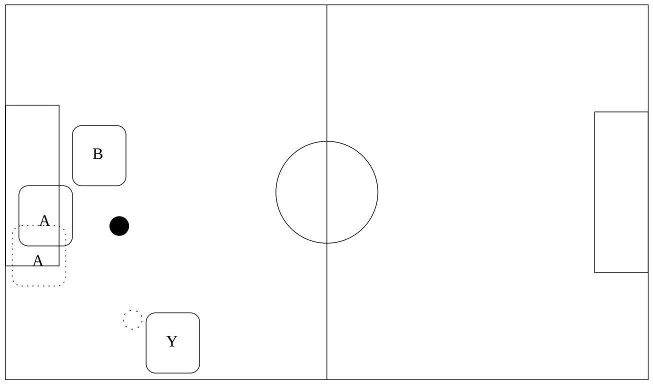


Figure 2: Graphical View of Example Case

gions. Within each partition of the case base, cases are stored in unordered lists. To retrieve a similar past case, a new case is compared to all past cases having the same ball region. A smaller distance between the past and current balls represents a better matching case. The current blocking position for the goalie is based on the blocking position of the most similar past case that had a successful outcome.

Once this simple prototype was implemented, the simulated goalie exhibited reasonable movement, but numerous problems and opportunities for improvement immediately became apparent. For one thing, we needed to consider additional factors, like whether or not another defender was in position to help the goalie. The importance weights accorded the various factors could also be fine-tuned. The case base needed enlargement to provide a wider range of past experiences. New cases could be obtained and added to the case base during play to account for differences among opponents. Failures could be analyzed to avoid repeating past mistakes. Perhaps, when a goal was scored against the goalie, the goalie had made the best possible move under the circumstances, but the opponent was just too good. This represents failure for the team, but can blame be assigned to the goalie?

While this prototype encouraged us that CBR could work for the RoboCats, and improved our understanding of what needed to be done to make it work, we decided against using CBR to implement goalie positioning in our robots. There were multiple reasons for this decision. From a practical standpoint, building a really good case-based reasoner was going to take a lot of time and effort. In the meantime, our goalie robot had become the strongest part of the RoboCats team, making the time and effort better spent elsewhere. Furthermore, the goalie was functioning quite well using only a reactive schema in which it tracked the position of the ball and moved along the mouth of the goal accordingly. We were unlikely to see improvements in results proportional to our efforts. It became clear to us that that the opportunities for CBR in RoboCup were far greater at the team level, where the decisions that need to be made are more complex and require more deliberation.

2.3 Selecting Team Formations

Our next prototype differs from the work pioneered by AT Humboldt and from our preceding prototype, in that its focus is on making strategic decisions for the team as a whole, rather than for an individual player. In the RoboCats, the Meta Agent is responsible for selecting the team's formation and then assigning roles from that formation to the individual robots on the team. In this prototype, CBR is used to select the team's formation. The assignment of roles and the implementation of the roles assigned lie beyond the scope of this prototype.

Here, a case is based on a snapshot of the entire field. The positions of the players and the ball are derived from the raw vision data. From these positions, further discriminating features are derived, including whether the situation is offensive, defensive or transitional, and the appropriate short-term goals for the team. The high level goals the team may have are: *Get the Ball, Score a Goal,* and *Prevent the Opponent from Scoring*. Subgoals that help in achieving these goals are: *Control an Area, Prevent a Pass, Prevent a Shot, Shoot the Ball, Get in Position to Shoot, Send a Pass,* and *Receive a Pass*. The positions, goals and subgoals are all stored within the case, as the situation description.

Past cases also contain the formation that was selected to achieve the goals and subgoals. The formation facilitates teamwork and prevents players on the same team from inadvertently interfering with each other's actions. A formation is defined as a set of player roles. A player role specifies the region of the field that a player should cover and the task that the player should strive to accomplish there.

Sample cases are shown in Figures 3 through 8. In these figures, *A, B, C, D* and *E* represent our robots, while *V, W, X, Y* and *Z* represent our opponent's robots. Dotted lines indicate previous positions on the field. Figure 3 shows a situation that is clearly defensive, and so the associated high-level goal is to *Prevent the Opponent from Scoring.* The most important subgoal should be *Prevent a Shot*, since opponent *W* has the ball and is in a good shooting position. The next most important subgoal is to *Prevent a Pass* to player *Y*, since it too is in a good scoring position. Figure 4 shows the formation that was selected for this situation. Two players play back, with one of them on the ball. In this case, player *B* moves to get between the ball and the goal, while player *C* defends against moves by player *Y*.

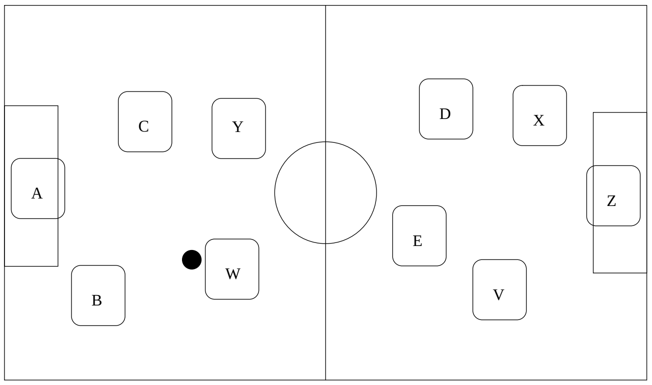


Figure 3: A Defensive Case

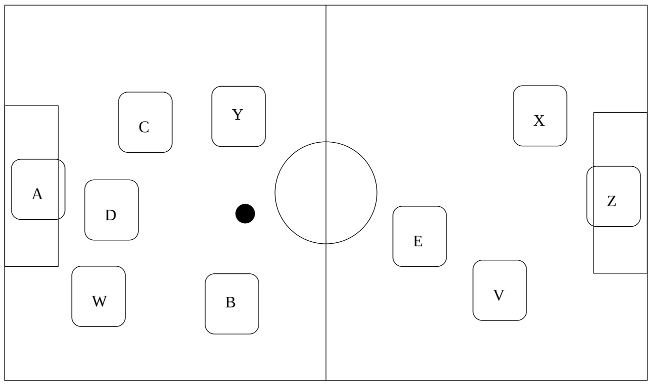


Figure 5: A Transitional Case

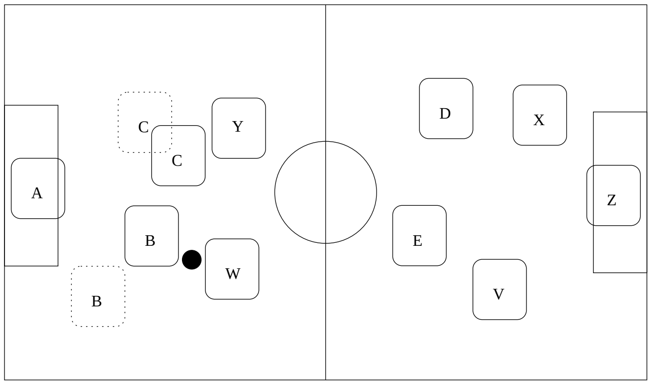


Figure 4: Selected Formation for the Defensive Case

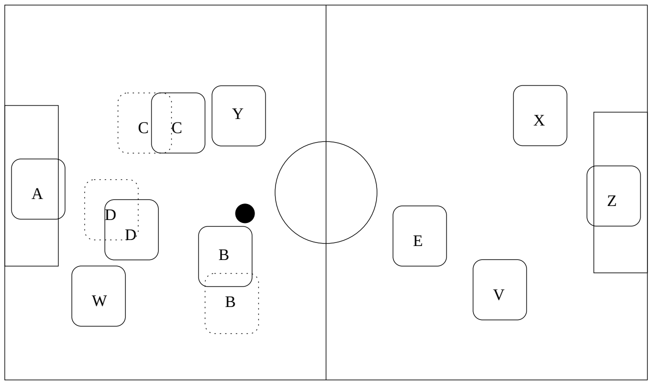


Figure 6: Selected Formation for the Transitional Case

Figure 5 shows a situation where the ball is loose and could be acquired by either team. Here, the high-level goal would be to *Get the Ball*. We would like to choose a formation that maximizes both defense, in case we are unable to get the ball, and offense, for a quick transition. The selected formation is shown in Figure 6. Here, one player is kept between the ball and the goal, while the closest player goes to get the ball. A third defender hangs back on the weak side of the field, i.e., the side of the field not containing the ball, to cover that region. In this case, player *D* is positioned between the ball and the goal, player *B* goes for the ball, and player *C* hangs back to block player *Y*.

Figure 7 shows a clearly offensive situation, where the high-level goal would be to *Score a Goal*. The chosen formation should maximize the likelihood of accomplishing the *Shoot the Ball*, *Send a Pass*, and *Receive a Pass* subgoals, while putting little emphasis on defense. Figure 8 shows the formation selected for this situation. The player in the best shooting position, in this case player *E*, stays up front and looks for shots and rebounds, taking them whenever possible. A second player, here player *D*, moves in, ready to receive a pass from player *E*, while covering its region. A third player, in this case player *B*, hangs back and tries to draw the opponent's defenders away from the goal.

When a formation must be determined during play, the current game situation is compared to the situation descriptions in the case base. The case base is partitioned into sections for defensive, transitional, and offensive cases. Since each game situation is classified as belonging to one of these categories, only cases in the associated partition must be considered. The formation stored in the most similar past case is returned, as the recommended formation, by the case-based reasoner to the Meta Agent. It is then the job of the Meta Agent to assign each role in the formation to a specific player and the job of the Player Agents to carry out the the roles they are assigned. While the ability of the Meta Agent to appropriately assign roles remains a significant unknown in the ultimate success of this approach, we are encouraged enough to extend this work for use in RoboCup 2003.

2.4 Recognizing Game States

Our third prototype recognizes game states as recurring patterns of behavior. It provides situation assessment that can be used by the Meta Agent and the Player Agents for planning team strategies and/or individual actions. Recognizing game states is a useful precursor to successfully accomplishing the tasks we tackled in our first two prototypes.

For this prototype, a case once again begins as a snapshot of the field. The vision data is first represented as the *x* and *y* coordinates, in millimeters, of the ball and each robot on the field. Symbolic feature value pairs are then derived from this data to describe the situation for each past and present case.

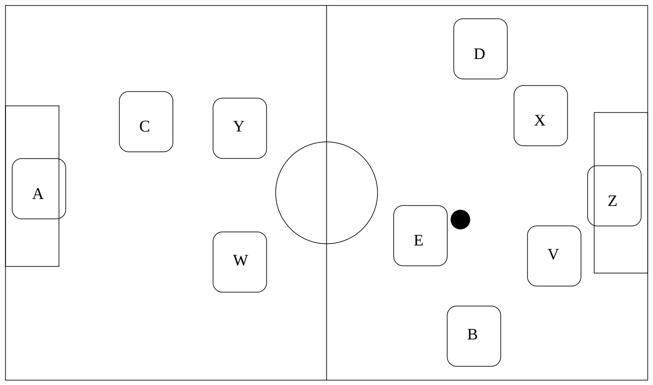


Figure 7: An Offensive Case

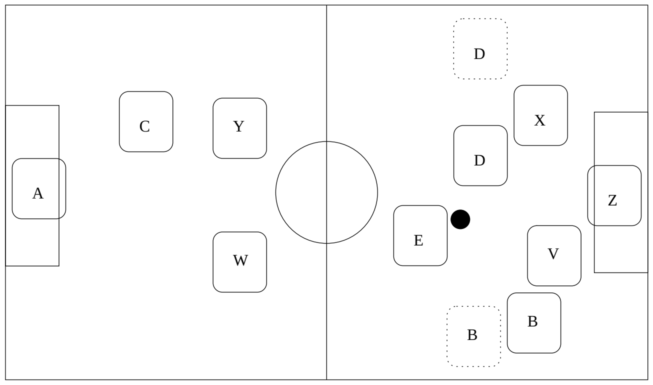


Figure 8: Selected Formation for the Offensive Case

Features stored include:

- *Defenders Back*, the number of defenders in the defensive zone
- *Attackers In*, the number of attackers in the offensive zone
- *Effective Defenders Back*, the number of defenders able to affect the play
- *Effective Attackers In*, the number of attackers able to affect the play
- *Attackers Covered*, the number of attackers being covered by defending robots
- *Effective Imbalance*, the imbalance between effective defenders and attackers, zero when teams are evenly matched
- *Ball Near Goal*, true if the ball is near the goal
- *Ball Carrier*, the team carrying the ball, unset if the ball is loose; and
- *Nearest to Ball*, the team nearest a loose ball, unset if one team possesses the ball.

Each past case also contains the game state, a characterization of play during the snapshot, such as *Man On* or *Two on One*. During play, the game state of a current case is determined by finding the best matching case in the case base and retrieving its state. A standard nearest neighbor algorithm is used to determine the best match. The overall flow of reasoning for this prototype is shown in Figure 9.

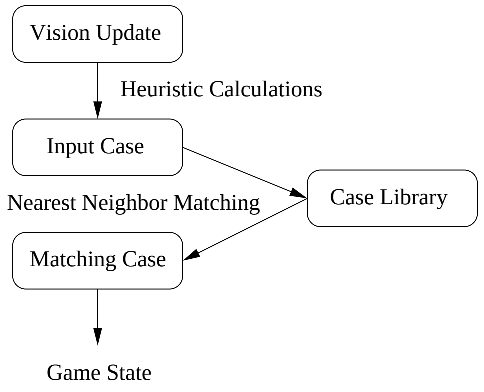


Figure 9: Flow of Reasoning in Game State Recognition

As an example, when the vision system reports the raw data shown in Table 2, the playing field would appear as shown in Figure 10. Table 3 shows the input case that would be derived from this data to describe this situation. The game state, as stored for a past case or retrieved for a current case, is *Possible Shot* for Player *E* or *Possible Pass* to an uncovered attacker, in this instance, from Player *E* to Player *D*.

This prototype is still a small, proof of concept, system. However, we plan to extend it for use in RoboCup 2003. Planned extensions include: deriving and incorporating additional descriptive features from the vision data; using two or more vision frames, instead of a snapshot, to capture information about the motion of the robots and the ball; and integrating this reasoner with other reasoning agents that can generate strategic plans based on the game state information provided.

Object	**x**	**y**
Ball	2060	478
Player *A*	215	1130
Player *B*	721	901
Player *C*	686	1366
Player *D*	1942	1449
Player *E*	1997	338
Player *V*	1435	2004
Player *W*	1713	882
Player *X*	2358	665
Player *Y*	1276	672
Player *Z*	2698	1061

Table 2: Sample Vision Data

3 Issues and Opportunities

We see both issues and opportunities in using CBR for physical agents in RoboCup and other dynamic, real-time environments. The first real-world problem we encountered is that mechanical, electrical and control issues slow the pace

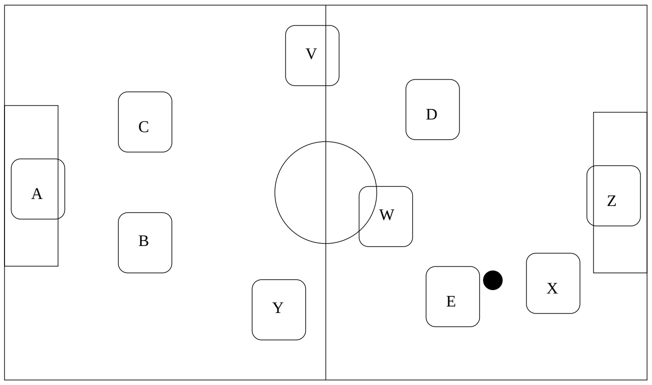


Figure 10: Graphical Case Representation

Feature	Value
Defenders Back	2
Attackers In	2
Effective Defenders Back	1
Effective Attackers In	2
Attackers Covered	1
Effective Imbalance	-1
Ball Near Goal	false
Ball Carrier	offense
Nearest To Ball	unset

Table 3: Case Derived from Sample Vision Data

at which we can add intelligence of any sort to our robots. The best intelligence is useless if the physical agents lack the ability to robustly and reliably do what they know how to do. This is analogous to the situation the first author finds herself in on those rare occasions when she plays soccer. As a dedicated soccer Mom, she *knows* how to play soccer very well. However, rusty actuators and weak batteries preclude her success on the field.

The second issue is that success, ie., winning the game, is not wholly attributable to having the best intelligence, even when all hardware is functional. As in human sports endeavors, a team may win by virtue of its brilliant game plan, its greater strength and stamina, the superior skills of a star player, or even sheer luck. Two teams may play well, but only one can win. This makes it difficult to assign credit or blame when things go well or badly. We need to develop and disseminate better guidelines and methodologies for evaluating the goodness of AI approaches in complex, real-world environments.

Third, it is not quick or easy to build a case-based reasoner. All of our cases have been built by hand, although it is possible to acquire subsequent cases automatically, once the structure of a case has been determined. The case is the basic knowledge representation for a case-based reasoner, and nontrivial knowledge representations require significant time and effort to develop. Cases must also be organized within the case base for efficient retrieval. Memory organizations can be fairly simple, when case bases are small or there are no real-time requirements, but substantial effort may be required to design and implement hierarchical memory structures that can speed case retrieval without sacrificing the ability to find the best case. Furthermore, retrieval metrics must be developed to determine case similarity. While we were able to use standard nearest neighbor matching as an overall framework, it was still necessary to determine the significant case features, how important each feature is for overall case similarity, and how to measure the degree of similarity among cases for each feature. When past solutions must be adapted to fit the present situation, it is usually necessary to determine and encode domain dependent adaptation strategies. This is often cited as the most difficult part of developing a case-based reasoner, and many implementors skirt the issue by avoiding case adaptation altogether.

Nevertheless, opportunity abounds. Our immediate plans are to integrate the CBR reasoners for selecting team formations and recognizing game states into the RoboCats robots in time for the next Robot Soccer World Cup, in Padua, Italy. We expect improvements over last year's performance due to greater team coordination, earlier recognition of impending threats, and more extensive inputs to our other reasoning agents.

We see an even greater opportunity for CBR in opponent modeling. Most RoboCup teams in the small size league deal with opponents by reacting to, rather than anticipating, opponent moves. Han and Veloso have noted that the ability to recognize, model and predict opponent behaviors is advantageous in RoboCup, because it enables teams to adapt their game strategies to effectively counter the opponent at hand [Han and Veloso, 1999]. They applied Hidden Markov Models (HMMs) to this task with positive results. HMMs are a good fit for recognizing opponent behaviors, because the opponent's strategy is hidden, but its moves from state to state provide clues about its likely next moves. However, Han and Veloso noted that there's a drawback, in that the complexity of behavior increases exponentially with the number of agents participating in that behavior. They limited their early work to behaviors involving a single robot and a ball. CBR is a promising alternate approach for this task that is not subject to the same scalability issue. The moves from state to state may be viewed as behavior patterns. A case can naturally represent a team behavior pattern as a whole. The advantage is that interactions among team members are implicit in the case, and need not be explicitly inferred by combining the behaviors of separate agents.

There is another aspect of opponent modeling that we are just beginning to explore with CBR. In human sports endeavors, a team may try to scout out the strengths and weaknesses of its opponent before a game, by watching the opponent on the field or by reviewing videos. A certain play or strategy may be more effective against one type of team than another. We are currently reviewing videos from RoboCup 2002 and constructing a case to represent each team. New opponents will be compared to those in our case base, and treated as we would treat the most similar known opponent. While much work lies ahead, this may well be the application that is most generalizable to other physical agents in adversarial, dynamic, real-time environments.

4 Future Work

It is easy to forget, in the heat of a RoboCup competition, that winning the game of soccer is not the only goal. Yet RoboCup, as an international research platform, has research, technological, and societal goals that transcend the soccer field. Approaches and technologies designed and developed for RoboCup are meant to be extensible to other problem domains, especially those with positive societal impact. A striking example of using RoboCup technology for the public good occurred when a team of search and rescue robots was deployed at Ground Zero in the aftermath of the World Trade Center attack in New York [Kahney, 2001]. The RoboCup Rescue robot league is an integral part of the international RoboCup competitions [The RoboCup Federation, 2003].

In this spirit, we have future plans to incorporate CBR into the Robotic Caregiver's Assistant (RCA), a team of robotic agents designed to provide in-home surveillance of Alzheimer's Disease (AD) patients. AD is a progressive brain disorder, marked by cognitive decline, forgetfulness, personality changes, bizarre behaviors, and impairment of the ability to complete simple daily tasks. At advanced stages of the disease, AD patients can not be left unsupervised, even for short periods of time, making life difficult for family caregivers. A primary concern for caregivers is patient safety, as patients are unable to respond appropriately in emergency situations, and are subject to hazards like falling, wandering away from home, getting lost, and starting fires [Whitehouse *et al.*, 2002].

Robotic teams could be used to assist these caregivers by continuously monitoring patients and alerting caregivers to potential dangers. As in robot soccer, teams of robots would cooperate to achieve a goal. Whereas in soccer the goal is to win the game, here the goal is to guarantee the patient's safety. As in robot soccer, there is an active adversary, which must be understood and thwarted. Whereas in soccer the adversary is the opposing team, here the adversary is any unsafe condition, whether caused by patient actions or stemming from the surrounding environment. As in robot soccer, the team's success depends on its ability to observe the adversary and to predict potential threats, in a dynamically changing environment, in real-time. CBR will be used to classify observed situations as dangerous or benign, to predict patient movements, and to project whether a patient's next move could lead to danger.

5 Related Research

While early work in CBR focused on standalone, single-modality systems, the current trend is to integrate CBR with other reasoning modalities in hybrid intelligent systems [Marling *et al.*, 2002]. For example, CBR has been integrated with rule-based reasoning, constraint satisfaction problem solving, model-based reasoning, genetic algorithms, information retrieval, STRIPS-style planning and hierarchical task network (HTN) planning. However, CBR is still seldom found as an integrated component of a hybrid deliberative/reactive robotic system.

The first robotic system to employ CBR, A Case-BAsed Reactive Robotic System (ACBARR), added CBR to a traditional reactive robot to improve its navigational capabilities [Ram *et al.*, 1992]. More recently, Fox has proposed using CBR as the basis of an overall planning architecture for a message delivery robot named RUPERT [Fox, 2000]. The idea behind RUPERT is that, to navigate successfully in a real world environment, a message delivery robot needs both reactive behaviors and high level plans. CBR is used to enable the robot to select an appropriate behavior or plan based on the current world situation. Both reactive behaviors and deliberative plans are stored as planning cases in a case library. Cases are indexed by the circumstances under which they were useful in the past. The robot analyzes the current world situation, and then searches the case library for the most similar stored situation. The most similar case is retrieved. Then the robot can apply the associated behavior or plan, without having to make an explicit choice of deliberating or reacting.

As previously noted, the use of CBR in RoboCup was pioneered by AT Humboldt, in the simulation league [Burkhard *et al.*, 1998; Gugenberger *et al.*, 1999]. In AT Humboldt's winning implementation, an individual player determined where to move next, based on previous moves made in similar game situations. Case retrieval nets were used by this team to enable real-time case retrieval, even in very large case bases [Lenz and Burkhard, 1996]. Researchers for AT Humboldt hypothesized early on that CBR would be useful for off-line learning of capabilities and for on-line learning of opponent behaviors [Burkhard *et al.*, 1998].

Despite the limited use of CBR in physical agents to date, it should be noted that CBR was originally proposed to facilitate the types of tasks physical agents may be called upon to perform. These include planning, adversarial reasoning, classification and interpretation, and prediction and projection [Kolodner, 1993]. Perhaps the earliest system to demonstrate CBR's potential for RoboCup was COACH [Riesbeck and Schank, 1989]. COACH planned strategic plays for American-style football teams.[1] It did so by storing plays, including moves and roles for specific players, and retrieving the best play to suit a current situation. It would adapt a retrieved play, when necessary, to account for differences in the past and present game situations.

6 Summary and Conclusions

This paper presents our work in developing case-based reasoners for the RoboCats, a team of five robots competing in the RoboCup small size league. We have developed three CBR prototypes, in simulation mode, for the tasks of positioning the goalie, selecting team formations, and recognizing game states. While our first prototype did not provide any advantage over positioning the goalie reactively, the other two prototypes enabled greater coordination of different team members working to accomplish mutual goals and earlier recognition of threats from opponents. We are currently integrating these two reasoners into our robotic team

[1]In American-style football, as in soccer, a team scores by moving a ball across a field against the efforts of opposing players. Unlike in soccer (which is also called football), the ball is more frequently moved by hand than by foot, and knocking over opponents is not only legal, but encouraged.

for use in the next Robot Soccer World Cup. We believe that, especially as robotic hardware becomes more robust and better methodologies are developed for evaluating robotic intelligence, CBR will become an important enabling technology. It is especially useful in situations where reactivity alone is insufficient to guarantee that appropriate actions occur within time constraints, and for capturing and considering the actions and interactions of multiple agents as a unified whole.

Acknowledgments

This work has been supported, in part, by the NASA CETDP grant "Resource Management for Real-Time Adaptive Agents" and by the Ohio University 1804 Research Fund. The authors gratefully acknowledge the support of Mrs. Beth K. Stocker, whose generosity enabled the RoboCats to travel to Fukuoka, Japan, to compete in RoboCup 2002, and is enabling the trip to Padua, Italy, for RoboCup 2003. The authors would also like to thank the entire RoboCats team, without whose contributions this work would not be possible.

References

[Birk *et al.*, 2002] A. Birk, S. Coradeschi, and S. Tadokoro, editors. *RoboCup 2001 : Robot Soccer World Cup V*. Springer, Berlin, 2002.

[Burkhard *et al.*, 1998] H.-D. Burkhard, M. Hannebauer, and J. Wendler. AT Humboldt — development, practice and theory. In H. Kitano, editor, *RoboCup-97 : Robot Soccer World Cup I*, Berlin, 1998. Springer.

[Fox, 2000] S. E. Fox. A unified CBR architecture for robot navigation. In E. Blanzieri and L. Portinale, editors, *Advances in Case-Based Reasoning: 5th European Workshop - EWCBR 2000*, pages 406–417, New York, NY, 2000. Springer.

[Gillen *et al.*, 2002] M. Gillen, A. Lakshmikumar, D. Chelberg, C. Marling, M. Tomko, and L. Welch. A hybrid hierarchical schema-based architecture for distributed autonomous agents. In *Proceedings of the AAAI Spring Symposium on Intelligent Distributed and Embedded Systems*, Menlo Park, CA, 2002. AAAI Press.

[Gugenberger *et al.*, 1999] P. Gugenberger, J. Wendler, K. Schröter, and H.-D. Burkhard. AT Humboldt in RoboCup-98 (Team description). In M. Asada and H. Kitano, editors, *RoboCup-98 : Robot Soccer World Cup II*, Berlin, 1999. Springer.

[Han and Veloso, 1999] K. Han and M. Veloso. Automated robot behavior recognition applied to robotic soccer. In *Proceedings of the IJCAI-99 Workshop on Team Behaviors and Plan Recognition*, pages 47 –52, 1999.

[Kahney, 2001] L. Kahney. Robots scour WTC wreckage, 2001. Wired News, World Wide Web site, http://www.wired.com/news/technology/0,1282,46930,00.html.

[Kolodner, 1993] J. Kolodner. *Case-Based Reasoning*. Morgan Kaufman, San Mateo, CA, 1993.

[Lenz and Burkhard, 1996] M. Lenz and H.-D. Burkhard. Case retrieval nets: Basic ideas and extensions. In G. Görz and S. Hölldobler, editors, *KI–96: Advances in Artificial Intelligence*, pages 227–239, Berlin, 1996. Springer.

[Marling *et al.*, 2002] C. Marling, M. Sqalli, E. Rissland, H. Munoz-Avila, and D. Aha. Case-based reasoning integrations. *AI Magazine*, 23(1):69–86, 2002.

[Ram *et al.*, 1992] A. Ram, R. C. Arkin, K. Moorman, and R. J. Clark. Case-based reactive navigation: A case-based method for on-line selection and adaptation of reactive control parameters in autonomous robotic systems. Technical Report GIT-CC-92/57, Georgia Institute of Technology, 1992.

[Riesbeck and Schank, 1989] C. K. Riesbeck and R. C. Schank. *Inside Case-Based Reasoning*, pages 291–318. Erlbaum, Hillsdale, NJ, 1989.

[The RoboCup Federation, 2002] The RoboCup Federation. Robocup official site, 2002. World Wide Web site, http://www.robocup.org/02.html.

[The RoboCup Federation, 2003] The RoboCup Federation. Robocup-rescue official web page, 2003. World Wide Web site, http://www.r.cs.kobe-u.ac.jp/robocup-rescue/.

[Wendler and Lenz, 1998] J. Wendler and M. Lenz. CBR for dynamic situation assessment in an agent-oriented setting. In D. Aha and J. J. Daniels, editors, *Case-Based Reasoning Integrations: Papers from the 1998 Workshop*, pages 172–176, Menlo Park, CA, 1998. AAAI Press.

[Whitehouse *et al.*, 2002] P. Whitehouse, C. Marling, and R. Harvey. Can a computer be a caregiver? In K. Haigh, editor, *Automation as Caregiver: The Role of Intelligent Technology in Elder Care: Papers from the AAAI Workshop*, Menlo Park, CA, 2002. AAAI Press.

A Real-Time Rover Executive Based On Model-Based Reactive Planning

M. Bernardine Dias

Carnegie Mellon University
5000 Forbes Ave.
Pittsburgh, PA 15213, USA
mbdias@ri.cmu.edu

Solange Lemai

LAAS/CNRS
7 ave. du Colonel Roche
31077 Toulouse cedex 4, France
slemai@laas.fr

Nicola Muscettola

NASA Ames Research Center
MS 269-2
Moffett Field, CA 94035, USA
mus@email.arc.nasa.gov

Abstract

This paper reports on the experimental verification of the ability of IDEA (Intelligent Distributed Execution Architecture) to effectively operate at multiple levels of abstraction in an autonomous control system. The basic hypothesis of IDEA is that a large control system can be structured as a collection of interacting control agents, each organized around the same fundamental structure. Two IDEA agents, a system-level agent and a mission-level agent, are designed and implemented to autonomously control the K9 rover in real-time. The system is evaluated in the scenario where the rover must acquire images from a specified set of locations. The IDEA agents are responsible for enabling the rover to achieve its goals while monitoring the execution and safety of the rover and recovering from dangerous states when necessary. Experiments carried out, both in simulation and on the physical rover, produced promising results. The rover successfully accomplished its goal while correctly responding to successive alarms. According to the preliminary results, we think that the system-level agent can achieve a 1-2Hz control rate on a 300MHz Pentium, adequate for slow-moving planetary rovers.

1 Introduction

Robotics space exploration requires autonomous control. While executing critical maneuvers or moving on rugged terrain the needed speed of control does not allow closing the loop with ground control due to large communication delays. Limited communication bandwidth and high personnel costs also increase the time and cost for recovering from on-board anomalies if large ground control crews are involved. The need to increase science output and operations safety while reaching for more ambitious and complex exploration goals strongly calls for more autonomous robots.

Some of the most autonomous space systems that have flown [Muscettola et al., 1998] or are preparing to fly [Chien et al., 2002] employ on-board automated planning systems. A planner receives goals either from the ground or from on-board task experts. The planner has access to a declarative model describing the necessary conditions for a plan to correctly achieve a goal and execute any supporting activities. The planner then builds a plan by interpreting the model through a standard planning engine, i.e., a search procedure that efficiently explores the possibly large number of ways to concatenate goals and supporting activities according to the model. This is done within the temporal and resource constraints intrinsic in the problem. Once a plan has been generated, it is read by a simple interpreter that issues appropriate commands to the performing system and monitors execution feedback returning from it.

Plan driven control is attractive in several respects. Perhaps the most important is the high level of assurance that it can deliver. The declarative model is essentially a constraint-based formal specification of the possible control behaviors of the system. In traditional flight software it is typical to manually translate this specification into the running code. Plan-based control instead eliminates this error-prone and difficult-to-validate development phase. Provided that the model correctly captures the physics of the devices and the desired control laws, the planning engine will guarantee the correctness of the control software. Of course, this argument relies on achieving a high level of assurance for the search engine. But reuse of the search engine without change across several applications subjects it to several cycles of rigorous testing, intrinsically increasing its reliability. Moreover, engine reuse also makes it economically feasible to use high-cost/high-reliability validation such as application of formal methods [Havelund et al., 2001].

However, so far planners are rarely used in on-board control systems for robots. When they are used, the planners are relegated to optimizing high-level task allocation over extended horizon while lower-level control is achieved with procedural execution [Simmons and Apfelbaum, 1998] or behavior-based control [Brooks, 1986]. This situation is partly due to a reaction to early attempts to build plan-based mobile control systems [Fikes et al., 1972] where planning was identified as a principal obstacle to the achievement of reactive behaviors. An important question, therefore, is whether it is possible to build planner-based core controllers that are fast enough to satisfy the reactive requirements of robotic controllers while fulfilling the high-assurance promise of plan-based computation.

This paper describes preliminary work in this direction. We describe the design and implementation of a rover controller that uses planning as the core-reasoning engine of a real-time executive. The control system has been demonstrated on the K9 rover test-bed (Figure 1) 0 at the NASA Ames Research Center. The tasks performed include some simple mission scenarios requiring the rover to take pictures with the on-board camera and recovering from simple faults such as excessive tilt and roll. The on-board executive was implemented using a general-purpose, planner-based distributed agent architecture, the Intelligent Distributed Execution Architecture (IDEA). The presented work demonstrates IDEA's viability for the implementation of real-time robotic controllers.

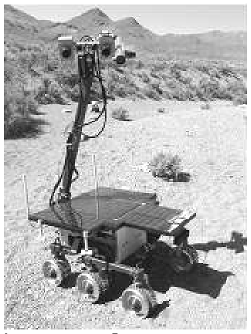


Figure 1 The K9 Rover

This paper is organized as follows. Section 2 gives a brief overview of IDEA agent architecture and describes how planning is integrated at the core of the execution cycle. Section 3 describes the test scenarios run on the K9 rover and how the scenario is modeled using separate IDEA agents. Section 4 reports experimental results while section 5 concludes the paper and discusses future work.

2 IDEA: Intelligent Distributed Execution Architecture

The most common organizational structure of autonomous control systems that have been used in practical applications is hybrid multi-layered, with several technologically diverse layers cooperating to achieve the robot's desired behavior. In mobile robotics, for example, a common layered controller separates between a low-level functional layer, often organized as a collection of controllers communicating according to a static routing map, and a high level decision layer using a procedural execution system [Nesnas et al., 2001]. Technological diversity among layers is problematic since each layer's machinery is described with a different computational model and supports different programming languages and methods without a clear mapping between them. This is problematic for two reasons. First, it increases the cost and difficulty of building complex autonomous controllers since a roboticist is supposed to thoroughly understand each computational model to be able to effectively program in it. Second, it increases the cost of validation and decreases the reliability of the software, since often the same information may need to be represented in two different ways in different layers. Moreover, lack of uniformity between layers makes the use of automated validation systems difficult.

The Intelligent Distributed Execution Architecture (IDEA) 0 postulates a different approach to the organization of complex autonomous controllers. The basic hypothesis is that a large control system can be structured as a collection of interacting control agents. Each atomic IDEA agent is structured in the same way and uses a model-based reactive planner as its core engine for reasoning. Each agent is required to operate with real-time guarantees. In fact, each agent has an intrinsic *execution latency*, a time quantum within which all computations needed to execute a "sense-plan-act" cycle must complete. If the execution latency is not respected, the IDEA agent is declared faulty and must be taken off-line. The execution latency allows bridging the perceived gap between AI-based methodologies to control and traditional control theory. In fact, the latency is directly mapped to a controller's sampling rate, the fundamental measurement of responsiveness in traditional control theory.

2.1 IDEA Control Agents

Figure 2 describes the core structure of an atomic IDEA agent. In comparison to earlier descriptions of IDEA [Muscettola et al., 2002], here we emphasize an organization that clearly isolates the services provided by IDEA and those that must be provided by third-party planning technology.

The agent communicates with external systems through a set of *goal registers*. At any point in time a register must contain an active goal describing the "interaction contract" with an external system. The content of the register always takes the form $P(\underline{i} \rightarrow \underline{s})$ where P is the name of a procedure, $\underline{i}$ is a (possibly empty) vector of input values and $\underline{s}$ is a (possibly empty) vector of return status parameters. When the goal is established, all arguments in $\underline{i}$ must be bound to some value $\underline{i_0}$ within the domain of possible values for $\underline{i}$. The contract terminates when either s is bound to a specific value, due to sensory feedback, or a timer associated to $<P, \underline{i_0}>$ expires. The latter allows procedures to be terminated by pre-emption in cases such as lack of response within a maximum allowable wait time. A subsystem interacting with the IDEA agent can be either *controlling* or *controlled*. It is controlled if the IDEA agent initially sets the value of the goal register with a new procedure and then waits for the controlled subsystem to set the status *s* or for the procedure timer to expire. It is controlling in the symmetrical case. A subsystem can be both controlling and controlled by interacting with the IDEA agent with different registers with different communication directions. Subsystems can be other IDEA agents or legacy software and hardware devices whose incoming and outgoing communications can be mapped into a finite set of goal registers maintained by the IDEA agent. The compositionality of the communication infrastructure allows the implementation of arbitrary distributed multi-agent control system structures.

Each goal register must behave according to a "timeline semantic". This means that at any point in time all goal registers must contain an active procedure. This, of course, cannot be satisfied when a procedure returns or must be terminated. In this case the agent goes through an execution cycle whose goal is to eliminate expired or returned procedures from goal registers and replace them with new procedures. The agent must perform this activity with a strict real-time guarantee, within the execution latency associated with the agent. The shorter the execution latency, the faster the IDEA agent can close the control loops in which it is involved.

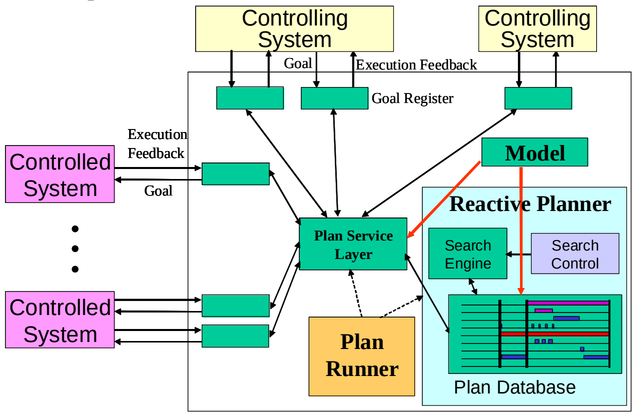


Figure 2 Structure of an IDEA agent

2.2 Quantifying Agent Reactivity

The module with the responsibility of starting and possibly aborting an execution cycle is the *Plan Runner*. The plan runner can only be activated at discrete times, synchronously with the agent's internal clock. The clock's granularity is the agent's execution latency. If a sensor value is received at time *t*, this will cause an execution cycle to start at time $k\lambda$ where λ is the agent's latency and $(k-1)\lambda \leq t < k\lambda$. Moreover, if the agent decides to start a new procedure during the execution cycle activated at time $k\lambda$, the procedure will be loaded in the goal register at a time τ, where $k\lambda \leq \tau < (k+1)\lambda$. Note therefore that in the worst case an IDEA agent's responsiveness, i.e., the maximum temporal distance between a stimulus (sensor value) and its response (the message announcing to the controlled agent that it should start a new procedure), is always 2λ. This permits precise quantification of the reactivity of a control agent, a measure that is usually elusive in control approaches based on planning or other Artificial Intelligence techniques.

2.3 Reactive Planning

The core reasoning in an IDEA agent is performed by the *Reactive Planner*. During an execution cycle, the reactive planner has the responsibility of determining the procedures with which expired goal registers should be loaded. The reactive planner explicitly represents histories for the agent's timelines in a *Plan Database*. These timelines describe past and future evolution of several entities: the content of each goal register (either incoming or outgoing), auxiliary state variables describing hypotheses on non-observable state of external systems, and state variables representing internal IDEA agent state implementing its control law. In the reference implementation of an IDEA agent, the planner uses a heuristic search procedure guided by search control rules programmed in an appropriate search control language. The planner conducts the search by continuously consulting a *Model*, i.e., a description of how procedures can follow each others on timelines and hence in goal registers. The model also describes in which way start and end of procedures can synchronize in all legal plans (see Section 3.3 for an example). By directly interpreting a declarative model, we believe that an IDEA agent can achieve higher levels of assurance than procedural approaches to plan execution and control.

2.4 The "Planning Bottleneck" Problem

The IDEA architecture supports several mechanisms for addressing the "planning bottleneck" problem that in the past has led to the summary dismissal of planning as a core control technology.

First of all, note that the architecture assumes the existence of a central plan database for each agent. It is possible for an agent to have several processes, besides the reactive planner, manipulate the plan database. Some of these processes can have the responsibility to build sections of plans over extended periods of time in the future, usually with the goal of "optimizing" some quality criteria. These processes operate at lower priority than the reactive planner and are controlled by the plan runner through goal registers, i.e., with the same coordination protocol used with external systems. Therefore, as long as the planning horizon over which the deliberative planner is working never intersects the current execution time, deliberative planning can operate in parallel with reactive execution and does not affect the reactivity of the agent.

The reactive planner itself may want to operate over planning horizons that are longer than the minimum possible one (one latency interval starting at the current execution time). However, the length of this horizon and the complexity of the model determine the worst case cost for solving a reactive planner problem and therefore determine the agent's latency. Vice versa, if the latency is bound by some characteristics of the controlled subsystems, one can deduce strict limits to the planning horizon as a function of the complexity of the model.

Reducing the planning horizon will cause the agent to be more reactively myopic which may require compiling more information in the model's "control law" timelines. Alternatively more extensive deliberative planning may be needed in advance, for example by explicitly representing contingency branches. This will allow the reactive planner simply to select an action among those cached in the plan database rather than having to synthesize a new plan from scratch every time.

Another way to tune the performance of an IDEA agent is to select a plan database/planning technology with the appropriate expressivity/performance tradeoff. For example, when it is important to reason about time, resources and bound uncertainty, then it could be appropriate to use constraint-based temporal planning technologies such as the one employed in the Remote Agent on-board planner. However, if the model matches an asynchronous discrete event control system, then a propositional representation and fast propositional incremental planning may be better suited to the task and achieve better performance. The IDEA architecture supports the use of different planning technologies by providing a standardized interface, the *Plan Service Layer*, between the planner and the goal register. Different planning technologies can be used as long as they can support a standard set of methods provided by the plan service layer. Also, an appropriate mapping must be defined between the modeling infrastructure of IDEA and the internal modeling of different plan database technologies.

In summary, the IDEA architecture provides an implementation of a set of basic services for building agents (goal registers and their input/output communication protocols, the plan runner, the plan service layer, the model) that we believe will be applicable across a wide variety of agents at multiple levels of abstraction in an autonomous control system. The proof of whether or not this goal can be achieved depends both on theoretical analysis and on experimental validations, such as the one reported in this paper.

3 A rover controller using IDEA

As an initial step towards validating IDEA, we have designed and implemented an IDEA controller for the K9 rover (Figure 1). The K9 rover is a six-wheeled, solar-powered rover complete with a manipulator. K9's mechanisms are a clone of those of the "FIDO" (Field Integrated Design and Operations) rover developed at JPL[Schenker et al., 1998]. The rover's avionics, instrumentation, and its autonomy software were developed at NASA Ames.

The rover carries a variety of instruments on board, including a compass, an inertial measurement unit and three pairs of monochromatic cameras (WideEye and 2 pairs of HazCams) used for navigation and instrument placement. Other instruments are mounted on an articulated arm that allows their precise placement for contact science. The WideEye stereo pair consists of a stereo pair of CMOS cameras mounted on a 10.93 cm baseline. Like the WideEye cameras, the front and rear HazCam stereo pairs consist of stereo pairs of CMOS cameras mounted on a 10.8 cm baseline. The rover also carries a pair of high-resolution, color stereo cameras (HawkEye), which consists of a stereo pair of high resolution multi-spectral cameras spaced on a 27.9 cm baseline, and the CHAMP, an arm-mounted, focusable microscopic camera developed at the University of Colorado, Boulder. The WideEye and HawkEye camera pairs are fitted on a Pan-Tilt unit. Our goal is to control this rover and its instruments via an IDEA controller.

In this section, we present the structure of the IDEA controller and its mapping to low-level rover control software. We then describe the test scenario and the models used by each IDEA agent to support this application. The scenario and the models have been tested in simulation and on-board the rover. Results are discussed in the next section.

3.1 Structure of the IDEA controller

Figure 3 depicts the mapping between the IDEA controller and the K9 controllers. The K9 controllers provide a functional layer of capabilities used by the IDEA controller. These capabilities include low-level commands – for instance the simple pan/tilt or camera commands – as well as some more complex behavioral commands, such as "drive to a position".

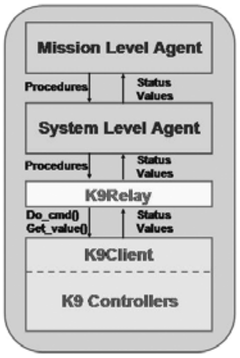


Figure 3 Mapping the IDEA agents to K9

Query functions can be used to obtain sensory information such as the rover's location, pitch/roll/yaw angles and the internal bay's temperature. The overall control software is composed by three subsystems organized in a three-layered hierarchy. The top layer of the hierarchy includes two IDEA agents: the system level and mission level agents. The bottom layer interacts with the system level agent according to the IDEA inter-agent protocol, although it is not implemented as an IDEA agent. The mapping is obtained through the K9Relay which behaves as a parser/decoder, translating the goals sent by the System level agent into the corresponding commands or information requests to the K9 controllers. We used CORBA as the underlying messaging infrastructure used to exchange goals and execution feedback between the IDEA agents and to exchange messages between the K9 controllers and the K9Relay.

3.2 Scenario

The IDEA control system has been tested on the following mission scenario. The rover must acquire images from several specified locations. A set of goals is sent to the rover, each consisting of a location and parameters for the camera and the pan/tilt unit. The rover decides in which order to accomplish these goals, monitors their execution and recovers from dangerous states.

Responsibilities have been assigned to the IDEA agents as follows. The mission-level agent receives goals (e.g. from the ground controllers) and decides on their best ordering using a deliberative planner. Execution of the plan at the mission-level sends one goal at a time to the system-level agent that is responsible for expanding lower-level activities, monitoring execution and planning recovery actions if necessary.

The system-level agent is responsible for monitoring rover safety while executing its plan. In particular, if safety limits for tilt and/or roll angles are exceeded, the system-level agent immediately stops the nominal execution, orders the rover to move backwards for a fixed distance, executes a turn in place by a fixed angle, and resumes execution of appropriate actions to achieve the goal. All of this is achieved through local reactive planning and plan execution.

3.3 Model description

The underlying planning technology used in both IDEA controllers is the EUROPA planning technology [Jonsson and Frank, 1999], a direct descendent of the Planner/Scheduler that was part of the Remote Agent [Jonsson et al., 2000]. The modeling language used for the agent models is the Domain Description Language (DDL) supported by EUROPA. Thus, designing a model is equivalent to defining a set of parallel timelines, sets of procedure types that can appear on each timeline and a set of constraints for each time interval over which a procedure can extend: temporal constraints between procedure intervals (also called compatibilities), duration constraints and parametric constraints that tie together all procedure variables (including the interval start time, end time, duration and input and status arguments of the procedure). An example of constraints' definition for the procedure *TempReadCompare* is shown in Figure 5.

Search control is implemented through heuristic rules used both by the reactive and deliberative planners. The rules prioritize subgoals that the planner should work on at each step of the search and prioritize slots on the timelines into which subgoals could be inserted. For the K9 controller, however, only a few heuristics were needed. They were used to prevent the Reactive Planner from trying to bind specific parameters, mainly the parameters corresponding to the return status, since their values are determined by the subsystem. Note that in principle it would be possible for the reactive planner to "guess" the return values of procedures. This is particularly important if the planner does look-ahead a few steps in the future or needs to develop contingent plans. In this case, the planner value of the return arguments would be checked with respect to the one actually obtained from the subsystem. If they do not match, then the reactive planner needs to modify the plan by accepting the true value returned by the subsystem and appropriately restructuring procedures. Our controller, however, is simple enough that the planner needs only to determine the next action without look-ahead and therefore can afford to leave the value of the return parameters unbound. This behavior is consistent with current approaches to procedural execution.

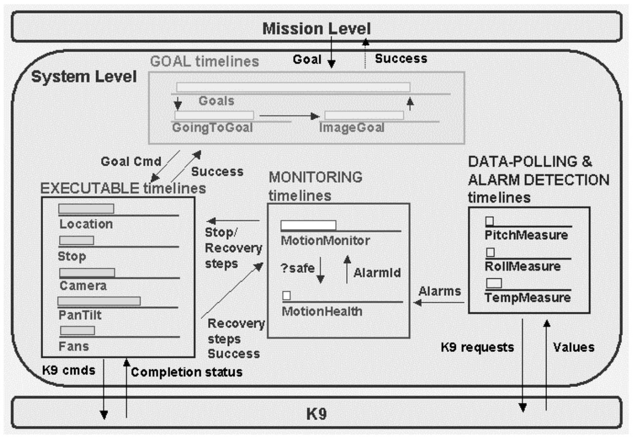


Figure 4 System level: interactions between timelines

Figure 4 depicts the interactions between the timelines defined in the System-Level's model. There are four types of timelines:

- The *Goal* timelines contain the goal sent by the Mission-Level and manage its completion.
- One timeline has been defined for each K9 component controlled by the agent: Location, Camera, Pan/Tilt unit, Fans. These *Executable* timelines contain procedures corresponding to the actual commands sent to the K9 controllers. For each command, a completion status is returned by the K9 controllers.
- To allow the monitoring of the rover safety, one *Data-Polling and Alarm Detection* timeline is defined for each monitored characteristic (pitch/roll angles, temperature, power…). These timelines contain procedures corresponding to information requests to the rover.
- For instance, at each execution cycle, a *PitchMeasure (→ ?alarm,?pitch,?pitch_rcvd)* goal is sent. The parameter *?pitch* is a status value returning the sensed pitch value, *?pitch-rcvd* an additional status parameter that determines whether the procedure terminated because a value was received for *?pitch* or because the procedure was pre-empted, and *?alarm* is another Boolean return status parameter. *?alarm*

and *?pitch* are linked by a constraint that sets *?alarm* to True if *?pitch* is greater than a predefined threshold. Once the Plan Runner has received and posted the value of *?pitch* in the plan database, the Reactive Planner applies the constraint, and a possible alarm is detected.

- For error recovery two other *Monitoring* timelines have been added to manage the different alarms and recovery steps. These timelines are especially useful with regard to the motion of the rover, as different motion alarms can occur at the same time and during the recovery actions. One timeline (MotionHealth) gives the state of the rover with an average frequency of 1/λ, where λ is the execution latency of the system-level agent. This is obtained by performing execution cycles at the maximum possible frequency 1/λ. If there is an alarm, the MotionHealth timeline identifies what type of alarm it is. Moreover, priorities can be defined between the different alarms. Each alarm corresponds to a specific sequence of recovery steps. The other timeline (MotionMonitor) is useful to manage the next recovery step to execute, depending on the evolution of the state of the rover. By means of compatibilities, the Reactive Planner will then insert the corresponding recovery procedures on the *Executable* timelines.

```
(Define_Compatibility
  (SINGLE((Rover_Class TempMeasure_SV))
     ((TempReadCompare(→?state_fan ?temp True))))
:duration_bounds [*temp_freq* *temp_freq*]
:parameter_functions
  (new_fan_state(*tempthreshold* ?temp ?state_fan))
:compatibility_spec
  (AND
     (meets (SINGLE ((Rover_Class Fans_SV))
              ((DeviceSetFanState (?state_fan→))))))))
```

Figure 5 Example of compatibility for the procedure TempReadCompare

Figure 5 gives an illustration of a simpler monitoring with an example of compatibilities for the procedure *TempReadCompare(→ ?state_fan,?temp,?temp_rcvd)* of the timeline TempMeasure. The temperature alarm detection is similar to the pitch case. Once the value of the temperature (status value *?temp*) has been received and posted by the plan runner *(?temp_rcvd* is set to True), the reactive planner applies the following constraints : the parameter-function *new_fan_state()* detects a possible alarm and sets the boolean *?state_fan* to True if necessary, then the compatibility *meets* inserts a command procedure *DeviceSetFanState(?state_fan→)* on the *Executable* timeline Fans. During the same control cycle a goal is sent to the K9Relay that translates into a direct command to the appropriate K9 low-level controller. This command finally turns the fan on. Note that *DeviceSetFanState* has an empty status vector. This is because we assume that the command will be executed in open loop without direct sensory feedback.

The system-level model contains only forward chaining compatibilities, since it is designed for a purely reactive agent, planning over a horizon covering only one execution latency ahead in reaction to new sensory information or new goals.

As stated before, the mission level agent receives a set of goals from the ground controllers. It uses deliberative planning to find the best ordering of the goals and sends one goal at a time to the System level agent for expansion and execution. The mission level monitors the completion of each goal and can replan if necessary.

The mission-level model contains three types of timelines. A set of *Internal* timelines is used by the deliberative planner to find the ordering of the goals. Deliberative planning is managed by means of a specific *Planner* timeline that contains Planning procedures whose parameters specify, notably, the start and end times of the planning horizon. The execution of such a procedure triggers the corresponding planning process. Finally, the plan resulting from deliberative planning (i.e. a sequence of goals) is put on a *Goal* timeline. This timeline is shared between the two agents. Its execution by the Reactive Planner at the mission level communicates one goal at a time to the System level and monitors the completion status returned back.

4 Results

We successfully controlled the K9 rover, via the System level agent such that the rover could detect successive situations where it exceeded safe thresholds in pitch and roll. The rover was able to use a fixed set of maneuvers to retract from these situations and search for alternate routes to accomplish the goal of arriving at a location specified by the mission-level agent and acquiring an image. Deliberative planning and interaction between the System level and Mission level agents were tested in simulation.

The performance of the System level agent's execution was further evaluated by monitoring the evolution of the elapsed time needed to complete a plan runner cycle and that of the fraction of the CPU used by the IDEA agent. Running an IDEA agent can require heavy use of the CPU, especially during deliberative planning at the mission level. The elapsed time taken by the plan runner cycle must not exceed the agent's execution latency. It mainly depends on the number of decisions made by the reactive planner during a cycle.

Both the mission and system-level agents were using EUROPA, a sophisticated constraint-based planning technology. In preliminary experiments on-board the rover (on a 300MHs Pentium), we noticed that the execution cycle time of the system-level agent monotonically increased during each test run. This was due to the fact that new decisions posted in the plan database at each reactive planner invocation made the constraint network grow monotonically. Thus each new invocation of the reactive planner required more and more time to propagate constraints throughout the plan database.

This problem was solved by noticing that system-level execution did not require remembering any of the past except for the currently executing procedure and the preceding one (to provide additional execution context). This allowed the implementation of a "forgetful" reactive planner, where a procedure is invoked before each application of reactive planning that "wipes out" the past from the plan database.

Figure 6 and Figure 7 show results obtained in simulation (on a 2,5GHz Pentium) using the forgetful reactive planner. We observe that the CPU usage never exceeds 30 % (whereas other runs not reported here show that 90% of CPU usage can occur during deliberative planning). The duration of the cycle is stable, the few peaks correspond to cycles where more decisions were made (reception and expansion of a goal, reaction to an alarm). The forgetful reactive planner has not been tested onboard the rover yet. But the results obtained during the previous experiments show that the duration of the cycle exceeds 0,5s only after 150s of test. Thus we believe that a control rate of 1-2Hz on a 300MHz Pentium is achievable for the system-level agent.

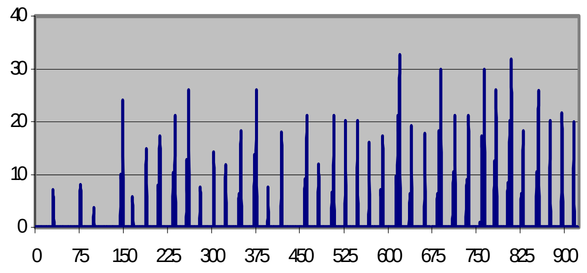


Figure 6 System Level agent: evolution of CPU usage (%) with time (s)

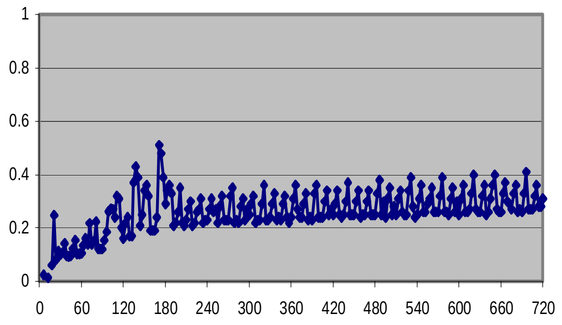


Figure 7 System Level agent: evolution of Plan Runner cycle duration (s) with time (s)

5 Conclusions and Future Work

In this paper we reported on preliminary experiments toward demonstrating the practical feasibility of a planner-based, multi-agent architecture for controlling mobility and remote sensing of a planetary rover. Much work remains to be done. To be viable for the limited computational resources available in flight systems, IDEA agents need to be as streamlined as possible. Any overhead in interpreting the model and searching for a reactive plan should be eliminated. We believe that much of this can be achieved by appropriately tuning the planner and increasing the efficiency of the planning technology used in each IDEA agent. In some cases, however, a purely search-based, "interpreted" approach may still be too slow. Therefore we plan to explore the feasibility of compilation schemes in which procedural executives satisfying the IDEA protocol are automatically generated from agent models. In this case the planner will still have a central role during system validation and, we believe, during the compilation phase. An interesting question that we will explore is characterizing the space/time tradeoff between a large but fast procedural expansion versus a more compact model encoding that is more slowly interpreted by a planner at run time.

Acknowledgments

This work was sponsored by the Automated Reasoning thrust of the NASA Intelligent System Program. We are grateful to the IDEA team members (Chuck Fry, Felix Ingrand, Rich Levinson, Chris Plaunt, and Baskaran Vijayakumar), the K9 team members (Maria Bualat, Rich Washington, Thomas Willeke, and Anne Wright) and Jeremy Frank of the EUROPA team for their support during the implementation of this work.

References

[Bresina et al., 2001] J. L. Bresina, M. Bualat, M. Fair, R. Washington, A. Wright, "The K9 on-board rover architecture", ESA Workshop on "On-board autonomy", 17-19 October 2001.

[Brooks, 1986] R. A. Brooks, A robust layered control system for a mobile robot, *IEEE Journal of Robotics and Automation*, 2:14-23, 1986.

[Chien et al., 2002] S. Chien, R. Sherwood, G. Rabideau, R. Castaño, A. Davies, M. C. Burl, R. Knight, T. Stough, J. Roden, P. Zetocha, R. Wainwright, P. Klupar, J. Van Gaasbeck, P. Cappelaere, D. Oswald, The Techsat-21 autonomous space science agent, *International conferences on Autonomous Agents and Multi-Agent Systems (AAMAS' 02)*, Bologna, Italy, 2002.

[Fikes et al., 1972] R. E. Fikes, P. E. Hart, N.J. Nilsson, "Learning and executing generalized robot plans", *Artificial Intelligence*, 3(4):251-288, 1972.

[Havelund et al., 2001] K. Havelund, M. Lowry, J. Penix. Formal Analysis of a Space Craft Controller using SPIN *IEEE Transactions on Software Engineering*, Volume 27, Number 8, August 2001.

[Jonsson and Frank, 1999] A. Jonsson, and J. Frank, A Framework for Dynamic Constraint Reasoning using Procedural Constraints, in *Workshop on Constraints in Control*, part of the *5th International Conference on Principles and Practices of Constraint Programming*, (CP99), 1999.

[Jonsson et al., 2000] A.K. Jonsson, P. Morris, N. Muscettola, K. Rajan, B. Smith, Planning in interplanetary space: theory and practice, in *Proceedings of the Fifth International Conference on Artificial Intelligence Planning Systems (AIPS'00)*, Breckenridge, Colorado, 2000.

[Muscettola et al., 1998] N. Muscettola, P. P. Nayak, B. Pell, and B. C. Williams. Remote Agent: To Boldly Go Where No AI System Has Gone Before. *Artificial Intelligence*, 103(1-2):5--48, 1998.

Muscettola et al., 2002] N. Muscettola, G. A. Dorais, C. Fry, R. Levinson, C. Plaunt. IDEA: Planning at the Core of Autonomous Reactive Agents, *WS On-line Planning and Scheduling, AIPS 2002*, Toulouse, France, pp. 49-55, 2002.

[Nesnas et al., 2001] I.A.D. Nesnas, R. Volpe, T. Estlin, H. Das, R. Petras D. Mutz, Toward Developing Reusable Software Components for Robotic Applications. *Proceedings of the International Conference on Intelligent Robots and Systems (IROS)*, Maui, Hawaii, 2001

[Schenker et al., 1998] P. S. Schenker, E. T. Baumgartner, R. A. Lindemann, H. Aghazarian, A. J. Ganino, G. S. Hickey, D. Q. Zhu, L. H. Matthies, Jet Propulsion Lab.; B. H. Hoffman, T. L. Huntsberger. New Planetary Rovers for Long-range Mars Science and Sample Return. *Intelligent Robots and Computer Vision XVI: Algorithms, Techniques, Active Vision, and Materials Handling*, SPIE Proc. Vol. 3522, pp. 215, Boston, MA, October 1998.

[Simmons and Apfelbaum, 1998] R. Simmons, D. Apfelbaum. A Task Description Language for Robot Control. *Proceedings of the International Conference on Intelligent Robotics and Systems (IROS)*, Vancouver, Canada, 1998.

Shared Activity Coordination

Bradley J. Clement and Anthony C. Barrett
Jet Propulsion Laboratory, California Institute of Technology
4800 Oak Grove Drive, M/S 126-347, Pasadena, CA 91109-8099 USA
voice +1-818-393-4729, fax +1-818-393-5244
{bclement, barrett}@aig.jpl.nasa.gov
Keywords: multiagent systems, planning, scheduling, real-time, communication

Abstract

While multiagent planning research has largely concentrated on distributing a planning problem or resolving conflicts among collaborative agents' plans, it has focused less on communication constraints, real-time issues, and negotation of self-interested agents. In domains where agents that interleave planning and execution have varying degrees of interaction and different constraints on communication and computation, agents will require different coordination protocols in order to efficiently reach consensus in real time. We briefly describe a largely unexplored class of real-time, distributed planning problems (inspired by interacting spacecraft missions), new challenges they pose, and a general approach to solving the problems. These problems involve self-interested agents that have infrequent communication but coordinate over joint activities and shared resources. We describe a Shared Activity Coordination (SHAC) framework that provides a decentralized algorithm for negotiating the scheduling of shared activities over the lifetimes of multiple agents, a soft real-time approach to reaching consensus during execution with limited communication, and a foundation for customizing protocols for negotiating planner interactions. We apply SHAC to a realistic simulation of interacting Mars spacecraft and illustrate the simplicity of protocol development.

1 Introduction

Interacting agents that interleave planning and execution must reach consensus on their commitments to each other before executing interdependent activities. When interleaving planning and execution, an agent adjusts its planned activities as it gathers information about the environment and encounters unexpected events. Interacting agents coordinate these adjustments to manage commitments with each other, but in the presence of communication constraints, reaching consensus on these commitments must be planned to avoid inconsistent execution. Demand for this kind of autonomous agent technology is growing for space applications. Autonomous spacecraft promise new capabilities and cost improvements in exploring the solar system. Spacecraft (and rovers) that explore other planets have intermittent, delayed communication with Earth, requiring that they be able to manage their resources and operate for long periods in isolation. The common approach to autonomous decision making is to place integrated data analysis, planning, and execution systems onboard the spacecraft.

In addition, there is a growing trend toward multi-spacecraft missions. Over forty multi-spacecraft missions have been proposed, including formation flying teams and over 16 planned missions to Mars in the next decade. These spacecraft will coordinate measurements, share data, and route data to and form Earth. Separate missions, such as those to Mars have their own budgets, experiments, and operations teams. As such, the spacecraft represent self-interested entities that benefit from collaborative interactions.

But, even a single spacecraft has multiple science instruments for executing different goals of different scientists, and different operations groups will have different areas of expertise over different subsystems for control. These different groups negotiate over mission plans in the same way that different Mars missions must collaborate over spacecraft interactions. Whether this negotiation is done on-board or on Earth, there is a distributed operations planning problem that benefits from automation. Both also have real-time aspects. On-board systems must plan safely over near- and long-term horizons, and ground systems must also converge on planss for daily, weekly, and lifelong mission operations. Ground planning also suffers from communication constraints. Scientists from different universities or opposite sides of the globe will intermittently provide inputs and respond on an irregular basis. A collaboration/negotiation system must be built around communication constraints to meet hard deadlines for coming to consensus on consistent operations plans.

In this work, we will briefly characterize this general problem in terms of activity interaction types and communication constraints and discuss its challenges. The field of multiagent planning has largely focused on fully cooperative planning and execution [Decker, 1995; desJardins and

The research described in this paper was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

Wolverton, 1999; Tambe, 1997; Grosz and Kraus, 1996; Clement and Durfee, 2000]. Market-based agent systems address near-term resource negotiation but have rarely addressed how near-term decisions affect longer-term goals. Multiagent systems built for Robocup Soccer competitions mainly address collaborative multiagent execution in an adversarial environment and have limited planning capabilities. These approaches do not adequately address real-time planning for self-interested agents.

We also present a framework for Shared Activity Coordination (SHAC). SHAC consists of an algorithm for continually coordinating agents and a foundation for rapidly designing and implementing coordination protocols based on a model of shared activities. In the same fashion that a real-time planning system must commit to actions to pass to an execution system, a real-time coordination system must additionally establish consensus on shared activities before they are executed based on communication constraints. Our ultimate goal is to create interacting agents that autonomously adjust their coordination protocols with respect to unexpected events and changes in communication or computation constraints so that the agents can most efficiently achieve their goals.

First we characterize a class of real-time, self-interested multiagent planning problems. Then we describe the shared activity model, the SHAC algorithm, and its interface to the planner. We also specify some generic roles and protocols using the SHAC framework that build on prior coordination mechanisms. In addition, we present an algorithm for determining how long a protocol will reach consensus under particular communication restrictions. Then we describe how our implementation of SHAC currently is used to coordinate the communication of two rovers and three orbiters in a simulated Mars scenario. We follow with future research needs revealed in this scenario and comparisons to related work.

2 Continual Coordination Problem

As mentioned before, agents that interleave planning and execution must commit near-term activities to the execution system while receiving feedback in the form of state updates and activity performance. One such continual planning system, CASPER (Continuous Activity Scheduling Planning Execution and Replanning) identifies the period when the planner commits activities to the execution system as a *commit window* [Chien *et al.*, 2000]. While the planner must resolve conflicts on activities before they are committed to execution, distributed planning agents must additionally reach consensus on team interactions before execution. As explored in the team plan model given by TEAMCORE [Tambe, 1997; Pynadath *et al.*, 1999], formalizations of Shared Plans [Grosz and Kraus, 1996], and coordination interactions of TAEMS [Decker, 1995], these interactions could include

- use and replenishment of shared resources,
- joint actions for achieving a mutually beneficial subgoal,
- choice of methods for achieving a team subgoal,
- participation and role assignments in a joint plan, and
- proposals and commitments of the above.

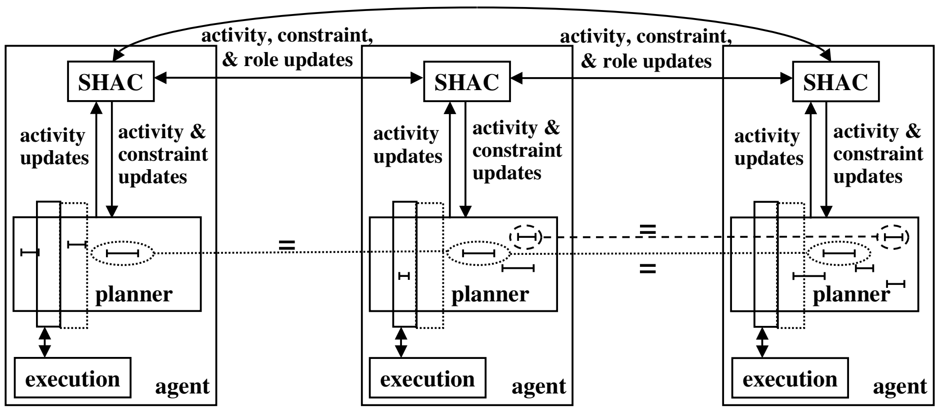
However, reaching consensus on these interactions is complicated when the agents can only communicate intermittently. Depending on the number of agents involved in a particular interaction, a consensus protocol may need to be initiated far in advance and negotiations settled far in advance of execution. Thus, for any particular set of interactions, a *consensus window*, within which the agents must limit negotiation and establish agreement, should be defined. For example, if three agents must negotiate a joint action in advance of execution but can only communicate pairwise in disjoint time windows, a consensus window must extend at least to cover windows connecting all three agents. Inside the consensus window, a simple protocol eliminating negotiation (such as all agree or reject) must be employed to guarantee consensus. Interactions beyond the consensus window can be negotiated with more elaborate protocols.

We informally describe the continual coordination problem because our approach is intended to be general to the capabilities of the individual planning and execution systems. The continual coordination problem can be specified generally as a continual planning domain and problem instance for each agent as well as communication constraints between agents. Continual planning problems have an evolving set of goals (as opposed to a single goal) and performance is measured as a function of the goals successfully achieved (executed) within a time horizon. The multiagent aspect of the problem is that interactions (such as those listed previously) create dependencies between the planning and execution systems, over which the agents must coordinate. The communication constraints can be time windows within which subsets of agents can communicate, the reliability of communication, the costs of communication (e.g. privacy), the bandwidth of channels, *etc.* These constraints determine how the agents can achieve consensus for a particular communications protocol.

Thus, many decisions must be made in the design of an efficient continual coordination system. What planning and execution systems are appropriate? What protocols should be used for negotiation and consensus? How should the consensus window be specified? Instead of providing a general solution to all of these questions, we describe an implemented framework for designing and evaluating protocols to negotiate interactions and establish consensus.

3 SHAC

Our approach, called Shared Activity Coordination (SHAC), provides a general algorithm for interleaving planning and the exchange of plan information based on shared activities. Agents coordinate their plans by establishing consensus on the parameters of shared activities. Figure 1 illustrates this approach where three agents share one activity and two share another. The constraints denote equality requirements between shared activity parameters in different agents. The left vertical box over each planner's schedule represents a *commit window* that moves along with the current time. A consensus window is shown to the right of the commit window, within which consensus must be quickly established before committing. Since consensus is hard to maintain when all agents can modify a shared activity's parameters at the same time, agents must participate in different coordination roles that specify which agent has control of the activity. As shown



activity, constraint, & role updates
activity, constraint, & role updates
SHAC
SHAC
SHAC
activity updates
activity & constraint updates
activity updates
activity & constraint updates
activity updates
activity & constraint updates
=
=
=
planner
planner
planner
execution
agent
execution
agent
execution
agent

Figure 1: Shared activity coordination

in the figure, SHAC interacts with the planning and execution by propagating changes to the activities, including their parameters and constraints on the values of those parameters.

SHAC continually coordinates by interfacing to a combined planning/execution system that responds to failures and state updates from the execution system. Our implementation interfaces with one such continual planning system, CASPER, mentioned in the previous section. Instead of batch-planning in episodes, CASPER continually adapts near and long-term activities while re-projecting state and resource profiles based on updates from sensors.

3.1 Shared Activities

The model of a shared activity is meant to capture the information that agents must share, including control mechanisms for changing that information. A shared activity is a tuple $\langle parameters,\ agent\ roles,\ protocols,\ decomposition,\ constraints\rangle$. The parameters are the shared variables and current values over which agents must reach consensus by the time the activity executes. The agent roles determine the local activity of each agent corresponding to the joint action. To provide flexible coordination relationships, the role activities of the shared activity can have different conditions and effects as specified by the local planning model. The shared parameters map to local parameters in the role activity.

For example, as shown by the shared activity instance in Figure 2, a data communication activity can map one agent to a `receive` role activity and another to a `send` role activity. Shared parameters could specify the start time, duration, transfer rate, and data size of the activity. The data size is depleted from the sender's memory resource but added to the receiver's memory. The agents could have separate power usages for transmitting and receiving. In this case the resources are not shared. Another shared activity could be the use of a common transport resource. Although one agent in an active transit role actually changes position, other agents in passive roles have local activities that only reserve the transport resource.

Protocols are mechanisms assigned to each agent (or role) that allow them to change constraints on a shared activity, the set of agents assigned to the activity, and their roles. In Figure 2, both agents use an argumentation protocol to negotiate the scheduling and attributes of the communication.

The shared decomposition enables agents to select different team methods for accomplishing a higher level shared goal. Specifically, the decomposition is a set of shared subactivities. The agents can choose the decomposition from a pre-specified set of subactivity lists. For example, a joint observation among orbiters could decompose into either (`measure`, `process_image`, `downlink`) or (`measure`, `downlink`).

3.2 Constraints

Constraints are created by agents' protocols to restrict sets of values for parameters (*parameter constraints*) and permissions for manipulating the parameters, changing constraints on the parameters, and scheduling shared activities (*permission constraints*). These constraints restrict the privileges (or responsibilities) of agents in making coordinated planning decisions. By communicating constraints, protocols can come to agreement on the scheduling of an activity without sharing all details of their local plans.

A parameter constraint is a tuple $\langle agent,\ parameter,\ value\ set\rangle$. The $agent$ denotes who created the constraint. Some protocols differentiate their treatment of constraints based on the agent that created them. For example, the asynchronous weak commitment algorithm prioritizes agents so that lower-priority agents only conform to higher-priority agent constraints [Yokoo and Hirayama, 1998]. Agents can add to their constraints on a parameter, replace constraints, or cancel them. A string parameter constraint, for example, can restrict a parameter to a specific set of strings. An integer or floating point variable constraint is a set of disjoint ranges of numbers. Scheduling constraints can be represented as constraints on a start time integer parameter. This is shown in Figure 2 where the rover restricts the start time of the communication between two eight minute intervals.

Permission constraints determine how an agent's planner is allowed to manipulate shared activities. The following permissions are currently defined for SHAC:

- parameters - change parameter values
- move - set start time

```
shared_activity communicate comm_id_12 {
  time start_time = 2004-302:09:30:00; // date
  int duration =  200;    // seconds
  int data_size = 25600;  // 25.6 Mbits
  real xmit_rate = 128.0; // Kbps
  int priority = 1;       // critical
  roles =
    receive by orbiter,
    send by rover;
  protocols =
    receive argumentation,
    send argumentation;
  permissions =
    receive (move, delete, xmit_rate),
    send (delete, data_size, priority);
  parameter_constraints =
    rover start_time = ([2004-302:09:30:00, 2004-302:09:38:00],
                        [2004-302:18:30:00, 2004-302:18:38:00]);
}
```

Figure 2: An instance of a shared communication activity between a rover and orbiter

- duration - change duration of task
- delete - remove from plan
- choose decomposition - select shared subactivity of an *or* activity
- add - add to plan[1]
- constrain - send constraints to other agents

In the communication example in Figure 2, the receiver is allowed to reschedule (move) the activity, delete it, or change the transmission rate. The sender cannot move the activity, but can delete it and change the requested size and priority.

3.3 Coordination Algorithm

The SHAC algorithm negotiates the scheduling and parameter values of shared activities until consensus is reached. Figure 3 gives a general specification of the algorithm. SHAC is implemented independently of the planner. Steps 1 through 3 are handled by the planner through an interface to SHAC. Step 4 invokes the protocols that potentially make changes to refocus coordination on resolving shared activity conflicts and improving plan utility. SHAC sends modifications of shared activities and constraints to sharing agents in step 5. In step 6, shared activities and constraints are updated based on changes received from other agents.

Ignoring coordination, a continuous planner must determine when it is appropriate to release activities to the execution system. In some cases, an activity involved in a conflict may either be released (requiring the planner to recover from potential failures) or postponed (to allow the planner to recover before a failure occurs). CASPER keeps a *commit window* (an interval between the current time and some point in the near future) within which activities cannot be modified and passes these activities to the execution system.

[1]This permission applies to a class of shared activities (i.e. an agent may be permitted to instantiate a shared activity of a particular class).

This interaction with the executive becomes more complicated when agents share tasks. SHAC must ensure that, when a shared activity is released, all agents release it while in consensus on the start time and other parameters of the task. Ideally the agents should establish consensus before the commit window. SHAC avoids changes in the commit window by keeping a *consensus window* that extends from the commit window forward by some period specific for the activity. As time moves forward, the windows extend forward. When a shared activity moves into the consensus window, the agents switch to the simple consensus protocol to try and reach consensus before the activity moves into the commit window.

4 Protocols

In general, protocols determine when to communicate, what to communicate, and how to process received communication. During each iteration of the loop of the coordination algorithm (Figure 3), the protocol determines what to communicate and how to process communication. A protocol is defined by how it implements the following procedures to be called during step 4 of the SHAC coordination algorithm for the shared activity to which it is assigned:

1. modify permissions of the sharing agents
2. modify locally generated parameter constraints
3. add/delete agents sharing the activity
4. change roles of sharing agents

The default protocol, representing a base class from which other protocols inherit, does nothing for these methods. However, even with this passive protocol, the SHAC algorithm still provides several capabilities:

joint intention A shared activity by itself represents a joint intention among the agents that share it.

mutual belief Parameters or state assertions of shared activities can be updated by sharing agents to establish consensus over shared information.

```
Given: a $plan$ with multiple activities including a set of $shared\_activities$ with $constraints$ and a $projection$ of
$plan$ into the future.
  1. Revise $projection$ using the currently perceived state and any newly added goal activities.
  2. Alter $plan$ and $projection$ while honoring $constraints$.
  3. Release relevant near-term activities of $plan$ to the real-time execution system.
  4. For each shared activity in $shared\_activities$,
       • if outside consensus window,
           – apply each associated protocol to modify the shared activity;
       • else
           – apply simple consensus protocol.
  5. Communicate changes in $shared\_activities$.
  6. Update $shared\_activities$ based on received communications.
  7. Go to 1.
```

Figure 3: Shared activity coordination algorithm

resource sharing Sharing agents can have identical constraints on shared states or resources.

active/passive roles Some sharing agents can have active roles with execution primitives while others have passive roles without execution primitives.

master/slave roles A master agent can have permission to schedule/modify an activity that a slave (which has no permissions) must plan around.

The following sections describe some subclasses of this abstract protocol, demonstrating capabilities that each protocol method can provide.

4.1 Argumentation

Argumentation is a technique for negotiating joint beliefs or intentions [Kraus *et al.*, 1998]. Commonly, one agent makes a proposal to others with justifications. The others evaluate the argument and either accept it or counter-propose with added justifications. This technique has been applied to teamwork negotiation to form teams, reorganize teams, and resolve conflicts over members' beliefs [Tambe and Jung, 1999]. It can also be used to establish consensus on shared activities.

A shared activity and associated parameter values are the proposal or counterproposal. Justifications are given as parameter constraints. A proposal is a change to a shared activity that does not violate any parameter constraints. A counterproposal may violate constraints. Protocol method 2 must be implemented to provide the parameter constraint justifications for proposals and counter-proposals. In order to avoid race conditions, protocol method 1 regulates permissions.

Argumentation method 1

- if this agent sent the most recent proposal/counterproposal
  - if planner modified shared activity
    - remove self's modification permissions
- else
  - give self modification permissions (e.g. move and delete)

Argumentation method 2

- if planner modified shared activity
  - generate parameter constraints describing locally consistent values

For example, one agent can propose an activity with a particular start time and add justifications in the form of all intervals within which the shared activity can be locally scheduled. Other agents can replan to accommodate the proposal and counter-propose with their own interval restrictions if replanning cannot accommodate others' constraints. If the agents cannot establish consensus before the consensus window, a higher ranking agent can mandate a time that benefits most of the agents. Of course, there are many variations on this example. Agents may be restricted because they are slaves or do not have constraint permissions to counter-propose.

4.2 Delegation

Delegation is a mechanism where an agent in a passive delegator role assigns and reassigns activities to different subsets of agents in active subordinate roles. The delegator and subordinate protocols only need to implement protocol method 3 to choose the subordinates sharing the activity.

Delegator method 3

- if $agent\ roles$ empty
  - choose an $agent$ to whom to delegate the activity
  - add ($agent$, subordinate) to $agent\ roles$

Subordinate method 3

- if cannot resolve conflicts/threats involving activity
  - remove self from $agent\ roles$

5 Defining Consensus Windows

Here we describe an algorithm for determining the shortest consensus window given a consensus protocol under particular communication constraints. Suppose agents have predetermined communication opportunity windows, such as orbiters having regular patterns of visibility to each other, a planetary surface, or Earth. We assume that communication is reliable with delivery time guarantees, that bandwidth is sufficient for coordination protocol communications, and that

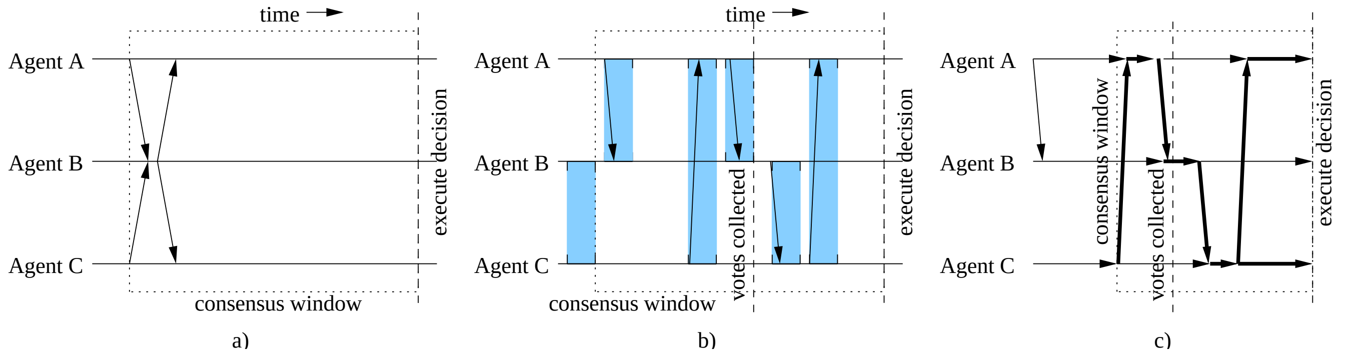


Figure 5: State-time diagrams of agent communication. a) no communication constraints; b) communication restricted to windows of opportunity; c) directed acyclic graph of information flow

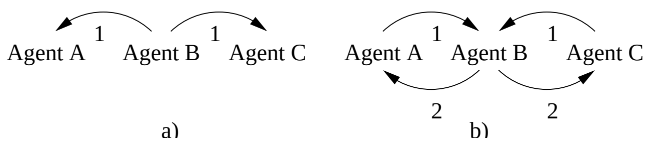


Figure 4: Information flow for consensus protocols: a) highest rank decides and b) voting or auction

the planner has worked out contention for power and memory resources required for communication.

Consensus protocols can be classified according to the flow of data and the computation time required to respond. Figure 4 shows examples for highest-rank-wins, voting, and auction protocols.[2] The voting protocol requires each participating agent to send a vote to a central agent, who (once all votes are received) can instantaneously determine the outcome that it must send to all participating agents. The space-time diagram in Figure 5a shows how voting reaches consensus for a decision when triggered upon entering the consensus window. However, this diagram assumes that agents can communicate at any time. Figure 5b shows the same protocol when pairs of agents can only communicate during specified periods (indicated by shaded regions). By walking through this diagram from left to right, the steps of the information flow diagram of Figure 4 are realized. In order to minimize the time to consensus, Agent C sends its vote/bid to B through A, and B sends the vote/auction outcome to A through C.

In order to determine how long the consensus window should be, we expand backwards through the state-time diagram according to the information flow diagram. To do this, we first construct a directed graph (G_c) as shown in Figure 5c from the execution time backwards, where each node is labeled by an estimated time value and an agent identifier. We also construct another directed graph (G_i) for the protocol's information flow, labeling edges according the state ordering (as done in Figure 4). Using these two graphs, the following algorithm determines the time to begin the consensus window by expanding a frontier of vertices backwards for each step (state) of the information flow diagram.

[2]We assume that the decision being voted or auctioned is priorly known by the agents.

```
Function Determine start time of consensus window
Input directed acyclic graph of communication opportunities
      $G_c(V_c, E_c)$, information flow diagram $G_i(V_i, E_i)$
Output $windowStart$
  • $windowStart$ = execution time (i.e. $\max_{v \in V_c}(v.time)$)
  • $state$ = $\max_{e \in E_i}(e.state)$
  • repeat
      – $flow = \{e \in E_i | e.state = state\}$
      – $synchTime$ = $windowStart$
      – for each $e_i \in flow$
          ∗ $frontier = \{v \in V_c | v.agent = e_i.dest \wedge$
            $v.time = synchTime\}$
          ∗ repeat
              · $E = \{e \in V_c | e.dest \in frontier \wedge$
                $e.source \notin frontier\}$
              · $e_c = \text{argmin}_{e \in E}(e.source.time)$
              · $frontier = frontier + \{e_c.source\}$
          ∗ until $e_c.source.agent = e_i.dest$
          ∗ $windowStart = \min(windowStart, e_c.source.time)$
      – $state$ = $state$ - 1
  • until $state$ = 0
  • return $windowStart$
```

The algorithm begins by setting $windowStart$ to the time of execution. It works backwards through the states of the information flow diagram, so $state$ is initially set to 2 for the voting protocol. The $flow$ set contains all of the edges in the flow diagram labeled 2. $synchTime$ is the time point between states and is used to populate $frontier$ with a vertice for each destination agent (agents A and C for the voting protocol) in the communication opportunity graph (G_c) at $synchTime$. After working back to state 1, $frontier$ only contains the vertice for Agent B just after the time that all votes are collected. The $frontier$ is greedily expanded backwards in the fashion of a tree spanning algorithm, iteratively adding vertices at time points closest to the frontier. Once the source agent of the information flow edge is reached, $windowStart$ records that latest time the source agent can send out information. This is done for each edge of the $flow$ for the same state since it can take longer for information to travel from different agents. Once all edges of one state are simulated, $windowStart$ contains the minimum time, representing the synchronization point ($synchTime$) for the prior

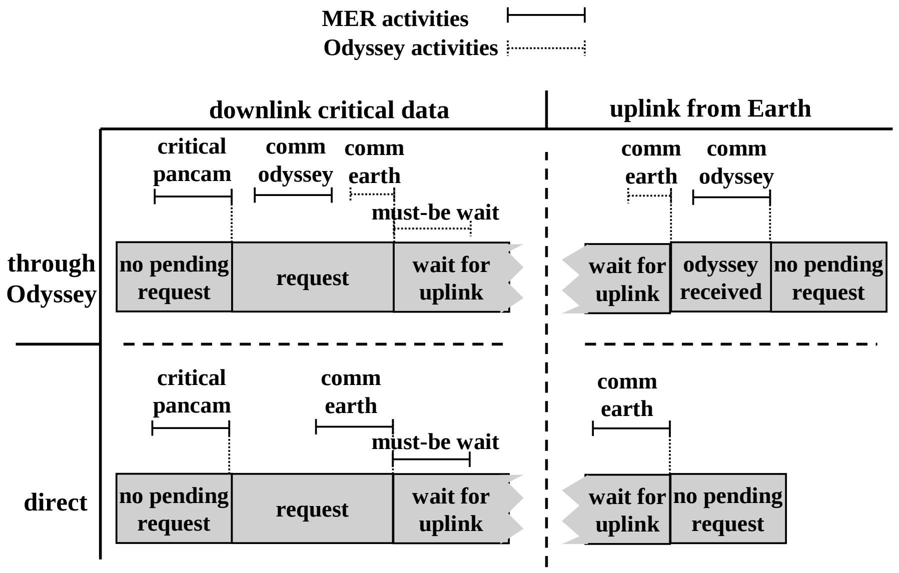


Figure 6: Downlink/uplink states for a rover

state to be simulated. Figure 5c shows thicker edges used to expand the frontiers of each state to find the latest possible consensus window start for the voting protocol. This is the actual flow of information for reaching consensus. Notice that the optimal start time is later than the example in Figure 5b.

The algorithm assumes that communication constraints are windows of communication and bounded delays on data transfer. The algorithm also assumes a simplified version of the information flow diagram that models the protocols. A possible expansion could include probabilistic information for unreliable transfer of data, and the algorithm could be adapted to give consensus windows with a probability of reaching consenus. Communications costs could also be included so that the algorithm could return optional consensus windows with different overall costs. While parallel information flow is modeled, the algorithm does not handle multiple hops within a single state. While somewhat limited, this approach can serve as a starting point for incorporating these extended capabilities.

6 Application to Mars Scenario

Now we describe how SHAC is applied to a simulated three-day scenario involving two Mars Exploration Rovers (MERs), the Mars Odyssey orbiter, Mars Global Surveyor, and the Mars Express orbiter. The delegation protocol described previously was subclassed for the rovers to assign and reassign the routing of images to the orbiters based on how quickly they can deliver the data to Earth. Different master/slave and active/passive roles are defined using permission constraints for the shared activities to implement a basic protocol for coordinating communication to and from Earth. Interactions over communication (once delegated) are between two agents, so the consensus window is defined to cover communication activities spanning two communication opportunities into the future. Once in the consensus window, the rover cannot redelegate activities unless the orbiter cannot resolve conflicts and must decommit. We intend to experiment with other protocols and consensus window definitions in this domain in our future work.

The MERs (MER A and MER B) and the orbiters can communicate with Earth directly, but the MERs can optionally route data through the orbiters, which talk with Earth at a higher bandwidth. The rovers need daily communication with ground operations to receive new goals. The rovers will often fail to traverse to a new target location and cannot proceed until new instructions come from ground operations. In this scenario both MERs must negotiate with the assigned orbiter to determine how to most quickly get a response from Earth after sending an image of their surroundings.

Each MER has a communication state shared with each orbiter that tracks when the image is generated, when it gets to Earth, and when the response from ground operations arrives to the rover. Shared activities for changing the state are shown for different routing options in Figure 6. The rover's activity for generating an image from its panoramic camera changes the state to `request` to communicate its need to downlink and receive an uplink. Activities for sending the image to Earth (either directly or through Odyssey) change the state to `wait for uplink` to indicate that the rover will then be waiting for the uplink. Ground operations needs a period of time to generate new commands for the uplink, so if the uplink is received by Odyssey, the state changes to `received` to indicate that now the rover can get the uplink from Odyssey. Once the rover receives the uplink, the state changes back to the normal `no pending request` state. Rover tasks (such as a traverse) need the uplinked data before executing, so it places a local constraint that shared state be `no pending request` during its scheduled interval. There are no shared resources although communication re-

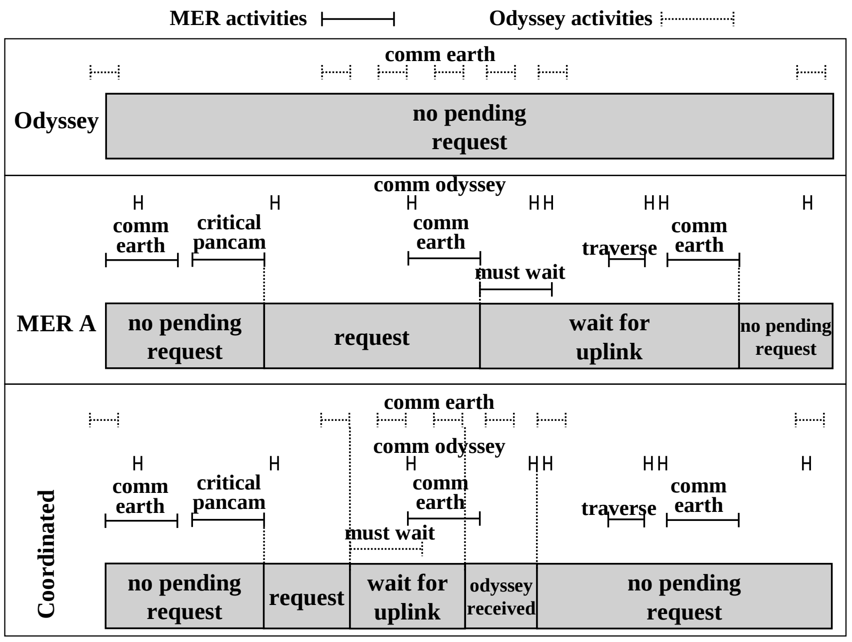


Figure 7: Downlink/uplink shared state for MER A. From top to bottom, Odyssey's initial view, MER A's initial view, and the common view after coordination.

quests from a MER have effects on many local resources of both the MER and the orbiter. All of the shared activities have active master and passive slave roles. The MERs and obiters both take the master role for activities labeled for them in Figure 6.

CASPER planners for each of the MERs and orbiters first build their three-day plans separately to optimize the timely delivery of priority weighted science data, resolving any local constraints on memory, power, battery energy, *etc*. The three-day schedules constitute over 900 tasks for each MER and over 1300 for each of the orbiters with 30 state/resource variables for each MER and 22 for the orbiters. Planning is slowed by a factor of 440 to account for differences between a desktop workstation and a radiation-hardened flight processor. Communication for coordination is restricted to times when the orbiters pass overhead. With the exception of Mars Express, the orbiters pass overhead once every eight hours. Because of its irregular orbit, Mars Express sees the rovers only once every 96 hours. Because of this, we actually used no consensus window for communication with Mars Express, thus, forcing the planners to resolve conflicts during image transmission.

When coordination begins, the planners send their communication requests to the other planners while optimizing their plans. Before these updates are received, the initial views of the shared uplink status are shown in Figure 7. The MERs begin with conflicts with their traverse tasks because the uplink has not yet been received from Earth. The coordination algorithm commands the planners to repetitively process shared task updates, replan to resolve conflicts by recomputing the shared state and modifying scientific measurement operations to adjust for the increased power and memory needs, and send task updates. After a minute and a half, MER A, B, and Odyssey agree on routing the downlink and uplink through Odyssey to get the uplinked commands in time for the traversal on different days. he resulting shared state is shown at the bottom of Figure 7. The planners reach consensus that coordination is complete and sleep while waiting for task updates.

Among other failed communication attempts, we triggered an anomaly in MER A's plan causing it to cancel its first day's tasks and shift the entire schedule forward a day. Before sending the updated shared tasks, replanning was issued to resolve local constraints to avoid propagating inconsistent state information to Odyssey. All conflicts were resolved in a few seconds except the traverse conflicts with a `wait` state. Then MER A sends a task update to restart coordination. Coordination completes in less than a minute with data again being routed through Odyssey.

While we have only experimented with simple protocols, this application of SHAC to the Mars scenario shows how planners can coordinate during execution while making minimal concessions to ideal plans and responding to unexpected events. In the next section, we discuss how SHAC builds on related work and discuss new research challenges for decentralized, coordinated planning.

7 Discussion and Related Work

Conflicts among a group of agents can be avoided by reducing or eliminating interactions by localizing plan effects to particular agents [Lansky, 1990], and by merging the individual plans of agents by introducing synchronization actions [Georgeff, 1983]. In fact, planning and merging can be interleaved [Ephrati and Rosenschein, 1994]. Earlier work studied interleaved planning and merging and decomposition in a distributed version of the NOAH planner [Corkill, 1979] that focused on distributed problem solving. More recent research builds on these techniques by formalizing and reasoning about the plans of multiple agents at multiple levels of abstraction to localize interactions and prune unfruitful spaces during the search for coordinated global plans [Clement and Durfee, 2000].

DSIPE [desJardins and Wolverton, 1999] employs a centralized plan merging strategy for distributed planners for collaborative problem solving using human decision support. Like our approach, local and global views of planning problem help the planners coordinate the elaboration and repair of their plans. DSIPE provides insight into human involvement in the planning process as well as automatic information filtering for isolating necessary information to share. While our approach relies on the domain modeler to specify up front what information will be shared, SHAC supports a fully decentralized framework and focuses on interleaved coordination and execution.

In many ways this work is following the Generalized Partial Global Planning approach to using a mix of coordination protocols tailored for the domain [Decker, 1995]. SHAC offers an alternative framework for separating implementation of these mechanisms from the planning algorithms employed by specific agents. Unlike GPGP, SHAC provides a modular framework for combining lower-level mechanisms to create higher-level roles and protocols. Our future work will build on GPGP's evaluations of mechanism variations to better understand how agents should coordinate for domains varying in agent interaction, communication constraints, and computation limitations.

Finally, TEAMCORE provides a robust framework for developing and executing team plans [Tambe, 1997; Pynadath *et al.*, 1999]. This work also offers a decision-theoretic approach to reducing communication within a collaborative framework. Research is needed to investigate the integration of coordinated planning with robust coordinated execution.

An assumption commonly made in multiagent research is that agents will be able to communicate at all times reliably. In the Mars scenario, the spacecraft communicate with each other in varying time windows and frequencies, and the two MERs can never directly talk to each other. Establishing consensus on beliefs and intentions is impossible without certain communication guarantees [Mullender, 1995]. Understanding the communication properties that make consensus possible and the overhead for establishing consensus is critical for multiagent research.

8 Conclusion

We informally described a continual coordination problem and its challenges. SHAC addresses the problem as a general planner-independent continual coordination algorithm and a framework for designing and evaluating role-based coordination mechanisms. We described its capabilities and gave examples of higher-level mechanisms built on these capabilities. In addition, we have presented an algorithm for determining the time to reach consensus given a protocol and communications constraints. While our future work is aimed at evaluating the benefits of different protocols for different classes of multiagent domains, we validate our approach in coordinating five simulated spacecraft experiencing unexpected events.

References

[Chien *et al.*, 2000] S. Chien, R. Knight, A. Stechert, R. Sherwood, and G. Rabideau. Using iterative repair to improve the responsiveness of planning and scheduling. In *Proc. ECP*, pages 300–307, 2000.

[Clement and Durfee, 2000] B. Clement and E. Durfee. Performance of coordinating concurrent hierarchical planning agents using summary information. In *Proc. ATAL*, pages 213–227, 2000.

[Corkill, 1979] D. Corkill. Hierarchical planning in a distributed environment. In *Proc. IJCAI*, pages 168–175, 1979.

[Decker, 1995] K. Decker. *Environment centered analysis and design of coordination mechanisms*. PhD thesis, University of Massachusetts, 1995.

[desJardins and Wolverton, 1999] M. desJardins and M. Wolverton. Coordinating a distributed planning system. *AI Magazine*, 20(4):45–53, 1999.

[Ephrati and Rosenschein, 1994] E. Ephrati and J. Rosenschein. Divide and conquer in multi-agent planning. In *Proc. AAAI*, pages 375–380, July 1994.

[Georgeff, 1983] M. P. Georgeff. Communication and interaction in multiagent planning. In *Proc. AAAI*, pages 125–129, 1983.

[Grosz and Kraus, 1996] B. Grosz and S. Kraus. Collaborative plans for complex group action. *Artificial Intelligence*, 86:269–358, 1996.

[Kraus *et al.*, 1998] S. Kraus, K. Sycara, and A. Evanchik. Reaching agreements through argumentation: a logical model and implementation. *Artificial Intelligence*, 104:1–70, 1998.

[Lansky, 1990] A. Lansky. Localized search for controlling automated reasoning. In *Proc. DARPA Workshop on Innov. Approaches to Planning, Scheduling and Control*, pages 115–125, November 1990.

[Mullender, 1995] S. Mullender. *Distributed Systems*. Addison-Wesley New York, 1995.

[Pynadath *et al.*, 1999] D. Pynadath, M. Tambe, N. Cauvat, and L. Cavedon. Toward team-oriented programming. In *Proc. ATAL*, 1999.

[Tambe and Jung, 1999] M. Tambe and H. Jung. The benefits of arguing in a team. *AI Magazine*, 20(4), 1999.

[Tambe, 1997] M. Tambe. Towards flexible teamwork. *Journal of Artificial Intelligence Research*, 7:83–124, 1997.

[Yokoo and Hirayama, 1998] M. Yokoo and K. Hirayama. The distributed constraint satisfaction problem: Formalization and algorithms. *IEEE Trans. on KDE*, 10(5):673–685, 1998.

Plays as Team Plans for Coordination and Adaptation [*]

Michael Bowling, Brett Browning, Allen Chang and **Manuela Veloso**
Computer Science Department
Carnegie Mellon University
Pittsburgh PA, 15213-3891

Abstract

Coordinated action for a team of robots is a challenging problem, especially in dynamic, unpredictable environments. We examine this problem in the context of robot soccer, a complex domain with multiple teams of robots in an adversarial setting. The adversarial nature creates a great deal of uncertainty, in both the opponent's behavior and capabilities. Traditional approaches focus on building reactive systems that use simple or even complex evaluation procedures for selecting team and individual robot actions given the state of the world. We introduce the concept of a *play* as a team plan, combining both reactive and deliberative principles. We introduce the concept of a *playbook* as a method for seamlessly combining many different team plans. The playbook provides a set of alternative team behaviors, and is the basis for our third contribution of *play adaptation*. We describe how these concepts were concretely implemented in the CMDragons robot soccer team, the first RoboCup robot team to adapt to its opponent during the game. We also show empirical results of the importance of adaptation in adversarial or other unpredictable environments.

1 Introduction

Coordination and adaptation are two of the most critical challenges for deploying teams of robots to perform useful tasks. These challenges become especially difficult in environments involving other agents, particularly adversarial ones, not under the team's control. In this paper, we examine these challenges within the context of robot soccer [6], a multi-robot goal-driven task in an adversarial environment. The robot soccer task is goal-driven and highly dynamic. The presence of adversaries creates significant uncertainty for predicting the outcome of interactions. This is particularly true if the opponent's behavior and capabilities are unknown a priori, as is the case in a robot soccer competition. As such, this task encapsulates many of the issues found in realistic multi-robot settings.

Despite this unpredictability, most robot soccer approaches involve single, static, monolithic team strategies (e.g., see robot team descriptions in [1].) Although these strategies entail complex combinations of reactive and deliberative approaches, they can still perform poorly against unknown opponents or in unexpected situations. With the uncertainty present in the task, such situations are common.

An alternative approach uses models of opponent behavior, constructed either before or during the competition [5], and then determine the best team response. The model may be used in a reactive fashion to trigger a pre-coded static strategy, or in a deliberative fashion through the use of a planner [7]. Although these techniques have had success, they have limitations such as the requirement for an adequate representation of opponent behavior. For a completely unknown opponent team, a prior model of their strategy is impractical.

Here, we take a novel approach based on observing our own team's effectiveness rather than observing the opponent. We replace a single monolithic team strategy, with multiple team plans that are appropriate for different opponents and situations, which we call *plays*. Each play defines a coordinated sequence of team behavior, and is explicit enough to facilitate evaluation of that play's execution. A *playbook* encapsulates the plays that a team can use. Each execution of a play from the playbook can then be evaluated and this information collected for future play selection. Successful plays, which may be due to weaknesses in the opponent or particular strengths of our team, are selected more often, while unsuccessful plays are ignored.

In Section 2, we overview our CMDragons'02 robot soccer team and the role of plays in the team's decision making. We then, in Section 3, describe plays and the play execution system in detail. In Section 4 we describe play adaptation with empirical results demonstrating its importance and effectiveness, and then conclude.

2 Overview

In this section, we briefly describe our CMDragons'02 robot soccer team, the basis for the work explored in this paper. This overview focuses on how the strategy system, described in Section 3, interacts with the system as a whole.

[*]This research was sponsored by Grants Nos. DABT63-99-1-0013, F30602-98-2-013 and F30602-97-2-020. The information in this publication does not necessarily reflect the position of the funding agencies and no official endorsement should be inferred.

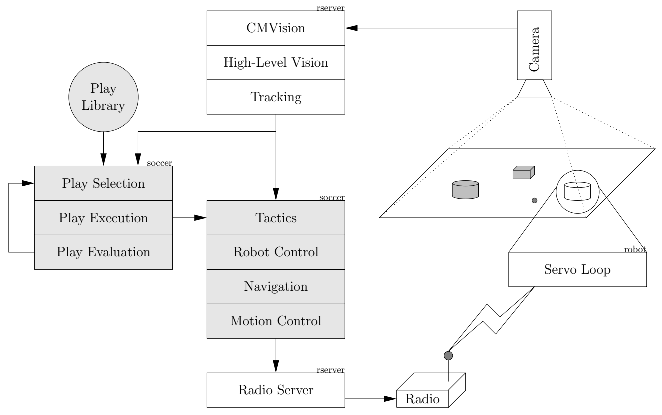


Figure 1: Overview of the CMDragons'02 team architecture.

The CMDragons are a team of small-size league (SSL) soccer robots that participated at RoboCup 2002. SSL robot soccer, part of the RoboCup initiative [6], consists of two teams of five robots that play soccer with an orange golf-ball on a 2.8m by 2.3m field surrounded by short, sloped walls using FIFA-like rules enforced by a human referee. Robots must conform to size and shape specifications, but no standard platforms exist. SSL is characterized by the allowance of cameras mounted above the field for shared global perception and additional off-field computation resources making a team as a whole autonomous rather than individual robots. SSL robots are typically fast, cruising at speeds of 1–2m/s while the ball moves at over 2m/s, and occasionally much faster. This makes SSL an environment that requires fast response, good long-term strategy, strong individual robot skills and capable multi-robot coordination, for a team to be successful.

Figure 1 shows the major components of the CMDragons system. The control loop, synchronized with image frames at 30Hz, consists of taking an image from the camera via the frame grabber and processing it, determining the control for each robot on the team, and sending these velocity commands using radio communication to the robots. Each robot operates local servo loops to enact its commands. Due to space limitations, we refer the reader to our earlier works [3] and [4], to describe the system in more detail. Instead, we focus on the tactics and strategy layers of the control software.

The tactics and strategy layers, the shaded regions in Figure 1, make up the bulk of the system. The tactics layer, encompasses individual robot skills. Here the notion of a *tactic* is synonymous with the term *Behaviors* often used in the literature. We use the terms tactics and strategy to reflect the differentiation between determining what the team members will do, and how each team member will do it. In practice, each robot executes a single tactic independently of the others each and every frame. The strategy layer provides the coordination mechanism and executes one instance for the entire team. Thus, the strategy layer must meld individual robot skills into powerful and adaptable team behavior. Here we focus on tactics, leaving discussion of strategy to the remainder of the paper.

Tactics are defined to be any behavior executable by a single robot. Table 1 shows the list of implemented tactics . Each tactic is highly parameterized and performs a complex, single robot task that itself may consist of many sub-components. Each tactic makes use of the robot control layer that maintains robot specific information persistent even after a robot's tactic changes. The layer transforms tactic commands into target points for navigation. The navigation layer produces short term, obstacle free, way-points for the motion control system using a fast randomized planner. These way-points are used by the motion control module to generate the actual velocity commands sent to the robot.

A tactic, therefore, is a complex interaction between low-levels of navigation and motion control and higher-level skill-based code. For example, consider the `position_for_deflection` tactic. The tactic itself determines the best location within a region for deflecting passes into the goal. This requires sampling points over the region and evaluating the deflection angles using a deflection heuristic. The best evaluated point is then fed into the navigation layer and in turn to the motion control layer to generate the actual commands necessary to drive the robot to the calculated position.

3 Play-Based Strategy

The main question addressed in this work is: "Given a set of effective and parameterized individual robot behaviors, how do we select each robot's behavior to achieve the team's goals?" This is the problem addressed by our strategy component, which is diagrammed by the left-most shaded components of Figure 1. Our team strategy utilizes the concept of a *play* as a team plan with multiple plays collected into a *playbook*.

3.1 Goals

The main criterion for team strategy is performance. A single, static, monolithic team strategy that maximizes performance, though, is impractical. Indeed, in an adversarial domain with an unknown opponent, a single optimal strategy is unlikely to exist. Therefore we have broken down the performance criterion into easier to achieve subgoals. The goals of a strategy system are:

1. Coordinates team behavior,
2. Executes temporally extended sequences of action,
3. Allows for special behavior for certain circumstances,
4. Allows ease of human design and augmentation,
5. Enables exploitation of short-lived opportunities, and
6. Allows on-line adaptation to the specific opponent.

The first four goals require plays to be able to express complex, coordinated, and sequenced behavior among teammates. In addition, plays must be human readable to make strategy design and modification simple. These goals also requires a powerful system capable of executing the complex behaviors the play describes. The fifth goal requires the execution system to also recognize and exploit opportunities that are not explicitly described by the current play. Finally, the sixth goal requires the system to alter its behavior over time.

Active Tactics	Non-active Tactics
shoot (Aim | Noaim | Deflect ⟨*role*⟩)	position_for_loose_ball ⟨*region*⟩
steal [⟨*coordinate*⟩]	position_for_rebound ⟨*region*⟩
clear	position_for_pass ⟨*region*⟩
active_def [⟨*coordinate*⟩]	position_for_deflection ⟨*region*⟩
pass ⟨*role*⟩	defend_line ⟨*p1*⟩ ⟨*p2*⟩ ⟨*min-dist*⟩ ⟨*max-dist*⟩
dribble_to_shoot ⟨*region*⟩	defend_point ⟨*p1*⟩ ⟨*min-dist*⟩ ⟨*max-dist*⟩
dribble_to_region ⟨*region*⟩	defend_lane ⟨*p1*⟩ ⟨*p2*⟩
spin_to_region ⟨*region*⟩	block ⟨*min-dist*⟩ ⟨*max-dist*⟩ ⟨*side-pref*⟩
receive_pass	mark ⟨*orole*⟩ (ball | our_goal | their_goal | shot)
receive_deflection	goalie
dribble_to_position ⟨*coordinate*⟩ ⟨*theta*⟩	stop
position_for_start ⟨*coordinate*⟩ ⟨*theta*⟩	velocity ⟨*vx*⟩ ⟨*vy*⟩ ⟨*vtheta*⟩
position_for_kick	position ⟨*coordinate*⟩ ⟨*theta*⟩
position_for_penalty	
charge_ball	

Table 1: List of tactics along with their parameters.

Notice that these goals, although critical to the robot soccer task, are also of general importance for the coordination of agent teams in other unpredictable or adversarial environments. We have developed a play-based team strategy, using a specialized play language, to meet these goals. In the following sections, we will describe the three major components of the play-based strategy engine: play specification using the play language, the play execution system, and the playbook adaptation mechanism used to autonomously alter team strategy to a specific opponent during a competition.

3.2 Play Specification

Plays are specified using the play language, which is in an easy-to-read text format (e.g., Tables 2 and 4). Plays use keywords, denoted by all capital letters, to mark different pieces of information. Each play has two components: *basic information* and *role information*. The basic information describes when a play can be executed (“APPLICABLE”), when execution of the play should stop (“DONE”), and some execution details (e.g., “FIXEDROLES”, “TIMEOUT”, and “OROLE”). The role information (“ROLE”) describes how the play is executed, making use of the tactics described above (see Section 2). We describe these keywords below.

```
PLAY Two Attackers, Pass

APPLICABLE offense
DONE aborted !offense

OROLE 0 closest_to_ball

ROLE 1
  pass 3
  mark 0 from_shot
  none
ROLE 2
  block 320 900 -1
  none
ROLE 3
  position_for_pass { R { B 1000 0 } ...
  receive_pass
  shoot A
  none
ROLE 4
  defend_line { -1400 1150 } ...
  none
```

Table 2: A complex play involving sequencing of behaviors.

Applicability. The APPLICABLE keyword denotes when a play can be executed. What follows the keyword is a conjunction of high-level predicates that all must be true for the play to be considered executable. Multiple APPLICABLE keywords can be used to denote different disjunctive conditions for when the play may be executed. This allows plays to effectively specify when they can be executed as a logical DNF of high-level predicates. In the example play in Table 2, the play can only be executed when the offense predicate is true. The offense predicate is a complex combination of the present and past possession of the ball and its field position. Predicates are easily added and Table 3 lists the predicates in use in our system.

A play’s applicability condition is very similar to an operator’s preconditions in classical planning. By constraining the applicability of a play we can design special purpose plays for very specific circumstances. Table 4 shows an example play that uses the in_their_corner predicate to constrain the play to execute only when the ball is in one of the opponent’s corners. The play explicitly involves dribbling the ball out of the corner to get a better angle for a shot on goal.

Termination. Unlike classical planning, the level of uncertainty in this task makes it difficult to predict the outcome of a particular plan. Although, a play does not have effects, it does have termination conditions. Termination conditions are specified by the keyword DONE followed by a result (e.g., aborted) and a conjunctive list of high-level predicates similar to the applicability conditions. Plays may have multiple DONE conditions, each with a different result, and a different conjunct of predicates. Whenever one of these DONE conditions are satisfied the play is terminated, and a new play must be selected. In the example play in Table 2, the only terminating condition is if the team is no longer on offense. In this case the play’s result is considered to have been aborted.

The results for plays are: succeeded, completed, aborted, and failed. These results are used to evalu-

`offense`	`our_kickoff`
`defense`	`their_kickoff`
`their_ball`	`our_freekick`
`our_ball`	`their_freekick`
`loose_ball`	`our_penalty`
`their_side`	`their_penalty`
`our_side`	`ball_x_gt` ⟨*X*⟩
`midfield`	`ball_x_lt` ⟨*Y*⟩
`in_our_corner`	`ball_absy_gt` ⟨*Y*⟩
`in_their_corner`	`ball_absy_lt` ⟨*Y*⟩
`nopponents_our_side` ⟨*N*⟩	

Table 3: List of high-level predicates.

ate the success of the play for the purposes of reselecting the play later. This is the major input to the play adaptation system which we describe in the next section.

There are two other ways in which plays can be terminated. The first is when the sequence of behaviors defined by the play are `completed`, which is described with the play execution system below. The second occurs when a play runs for too long without terminating. The timeout causes the play to terminate with an `aborted` result and a new play is selected. Thus, the team commits to a course of action, but if no progress is made due to unforeseen circumstances, another approach will be tried. The timeout period has a team configurable default value, however, a play may use the `TIMEOUT` keyword to override this default timeout limit (e.g. Table 4).

Roles. Roles are the active component of each play, and each play has four roles corresponding to the non-goalie robots on the field. Each role contains a list of tactics (also called behaviors) with associated parameters for the robot to perform in sequence. As tactics are heavily parameterized, the range of tactics can be combined into nearly an infinite number of play possibilities. Table 4 shows an example play where the first role executes two sequenced tactics. First the robot dribbles the ball out of the corner and then switches to the shooting behavior. Meanwhile the other roles execute a single behavior for the play's duration.

Sequencing implies an enforced synchronization, or coordination between roles. Once a tactic completes, all roles move to their next behavior in their sequence (if one is defined). Thus, in the example in Table 2, when the player assigned to pass the ball completes the pass, then it will switch to the mark behavior. The receiver of the pass will simultaneously switch to receive the pass, after which it will try to execute the shooting tactic.

Opponent Roles. Some behaviors are dependent on the positions of specific opponents on the field. Opponent roles are used to identify a specific opponent based on an evaluation method for the tactic to use. The example in Table 2, shows an opponent role defined using the `OROLE` keyword and the `closest_to_ball` method. Thus, the first role will try to mark the opponent closest to the ball away from the ensuing shot, after executing the pass.

Coordinate Systems. Parameters for tactics are also very general by allowing for a variety of coordinate systems in specifying points and regions on the field. Coordinates may be specified as absolute field positions or ball relative field positions. In addition, a coordinate system's positive *y*-axis can be oriented to point towards the side of the field that the ball is on, the side of field the majority of the opponents are on, or even a careful combination of these two factors. This allows tremendous flexibility in the specification of the behaviors used in plays and prevents unnecessary duplication of plays for symmetric field situations.

```
PLAY Two Attackers, Corner Dribble 1

APPLICABLE offense in_their_corner
DONE aborted !offense
TIMEOUT 15

ROLE 1
  dribble_to_shoot { R { B 1100 800 } ...
  shoot A
  none
ROLE 2
  block 320 900 -1
  none
ROLE 3
  position_for_pass { R { B 1000 0 } ...
  none
ROLE 4
  defend_line { -1400 1150 } ...
  none
```

Table 4: A special purpose play that is only executed when the ball is in an offensive corner of the field.

3.3 Play Execution

The play execution module is responsible for instantiating the active play into actual robot behavior. Instantiation consists of many key decisions: role assignment, role switching, sequencing tactics, opportunistic behavior, and termination.

Role assignment is dynamic, rather than being fixed, and is determined by uses tactic-specific methods. To prevent conflicts, assignment is prioritized by the order in which roles appear. Thus, the first role, which usually involves ball manipulation, is assigned first and considers all four field robots. The next role is assigned to one of the remaining robots, and so on. The prioritization provides the execution system the knowledge to select the best robots to perform each role and also provides the basis for role switching. Role switching is a very effective technique for exploiting changes in the environment that alter the effectiveness of robots fulfilling roles. The executor continuously reexamines the role assignment for possible opportunities to improve it as the environment changes. Although, it has a strong bias toward maintaining the current assignment to avoid oscillation.

Sequencing is needed to move the entire team through the list of tactics in sequence. When the tactic executed by the *active player*, the robot whose role specifies a tactic related to the ball (see Table 1), succeeds then the play transitions *each* role to the next tactic in their relative sequence. Finally, opportunistic behavior accounts for unexpected situations where a very basic action would have a valuable outcome. For example, the play executor evaluates the duration of time and potential success of each robot shooting immediately. If a robot can shoot quickly enough and with a high likelihood of success, it will immediately switch its behavior to take advantage of the situation. Thus, opportunistic behavior enables plays to have behavior beyond that specified explicitly. As a result, a play can encode a long sequence of complex behavior without encumbering its ability to respond to unexpected short-lived opportunities.

Finally, the play executor checks the play's termination criteria, the completion status of the tactics, and the incoming information from the referee. If the final active tactic in the play's sequence of tactics completes then the play terminates as completed. If the game is stopped by the referee for a goal, penalty, or free kick, the play terminates. The outcome of the play depends upon the condition. Foals and penalty kicks result in a success or failure as appropriate. A free kick is results in a completion or an abortion as appropriate.

3.4 Play Selection

The final detail of the playbook strategy system is the mechanism to select plays, and adapt that selection given experience. Our basic selection scheme uses the applicability conditions for each play to form a candidate list from which one play is selected at random. To adapt play selection, we modify the probability of selecting a play using a weighting scheme. In the next section, we describe this scheme and present experimental results showing the usefulness of a playbook approach and the effectiveness of adaptation.

3.5 Implementation

Here we briefly, due to space considerations, describe our implementation of the play system. Firstly, let us consider the larger issues of how plays fit in within the larger framework. Within the system architecture there are two main control paths (see figure 1). The primary control path flows from vision, through tactics, and to the robots. This path must handle information in a high bandwidth, low latency manner. The second control path flows from vision, through the playbook engine, then through tactics to the robot. In short, the playbook engine controls the robots through the tactics layer. Since the plays generate team behavior through the instantiation of tactics, which have a response time of seconds to tens of seconds, it is not strictly necessary for the playbook engine to operate with high bandwidth, which in CPU limited implementations can be useful. However, in practice, the playbook executor operates at frame rate to ensure that its latency, or response time, when reacting to changing situations is kept as small as possible. This is a critical issue to ensuring real-time team response. Moreover, great effort is exerted to ensure that all algorithms process within a single frame time (33ms). Hence, once accounting for the unavoidable latency incurred through using vision, the tactics and plays work in harmony to produce a new robot response within a single frame. Overall this produces a total system latency of 100ms when responding to events. Of course, the robot response times (e.g. driving to intercept a moving ball) then form an additional latency which is on the order of seconds usually.

The playbook engine consists of a number of interacting C++ classes. At the base are the *PlayRole*, *Play*, and *PlayBook* classes which encapsulate the data and methods for roles, plays and the playbook, respectively. A specialized class, called *PlayAscii* supports the reading of ASCII play files. The key part in the chain though, is the *PlayExecutor* class. This class is responsible for selecting plays, controlling the flow of play execution, and then modifying the play's weights based upon its performance.

The operations of the executor are as described in this section. There are two important aspects to help understand its implementation. Firstly, tactics are sub-classed from a base *Tactics* class. Tactics parameterization, as has been described, occurs through the constructor mechanisms. Thus, the play engine operates by creating with appropriate parameters and destroying tactics, which then in turn take generate robot actions. In addition each tactic comes armed with a range of evaluation functions, which the play executor makes use of in order to assign robots to roles. It is important to note here that tactics themselves are complex mechanisms involving predictive calculations, navigation planning, motion control and in some cases manipulation.

The second aspect to understanding the play executor is that its operation is greatly simplified by the use of predicates which are in turn derived from the output of the tracking system. The latter is encapsulated in the precociously named *World* class. Through Kalman filtering, and higher level processing of the output of the visual stream, the world model provides some fairly high level primitives for forming the predicates described in this paper. Furthermore, these predicates are reasonably, although not totally, free of noise thereby allowing reasonable encoding of plays using these symbolic predicates. As explained, these predicates form the basis for determining when a play is applicable, and also for detecting the success or failure of a play. The case where success or failure is determined by a goal being scored is detectable through the use of a referee box. This is essentially a laptop, which was introduced in 2002, which sends commands to each team computer translating the referee signals ('goal', 'free kick' etc.) into computer readable signals.

4 Playbook Adaptation

Playbook adaptation is the problem of adapting play selection based on past execution to find the dominant play or plays for the given opponent. We perform this adaptation using the execution outcomes of past selected plays. In order to facilitate the compiling of past outcomes into the selection process we associate with each play a weight, $w_{p_i} \in [0, \infty)$. These weights are then normalized over all the set of all applicable plays, A, to define a probability distribution,

$$Pr(\text{selecting } p_i) = \frac{w_{p_i}}{\sum_{p_j \in A} w_{p_j}}.$$

Playbook adaptation involves adjusting the selection weights given the outcome of a play's execution. An adaptation rule is a mapping, $W(\mathbf{w}, p_i, o) \rightarrow [0, \infty)$, from a weight vector, a selected play, and its outcome, to a new weight for that play. These new weights are then used to select the next play.

There are a number of obvious properties for an adaptation rule. All things being equal, more successes or completions should increase the play's weight. Similarly, aborts and failures should decrease the weight. In order for adaptation to have any effect, it also must change weights drastically to make an impact within the short time span of a single game.

The basic rule that we implemented for the RoboCup 2002 competition uses a weight multiplication rule, where each outcome multiplies the play's previous weight by a constant. Specifically, the rule is,

$$W(\mathbf{w}, p_i, o) = C_o w_{p_i},$$

where C_o is the constant associated with the particular outcome. We nominally set these to,

$$C_{\texttt{succeeded}} = 4 \qquad C_{\texttt{completed}} = 4/3$$
$$C_{\texttt{aborted}} = 3/4 \qquad C_{\texttt{failed}} = 1/4.$$

These weight values capture the basic properties described in the above paragraphs. Little work has gone into optimizing these weight selections. Since the opponent is unknown a priori, it seems difficult to know in advance what weight values are best. Rather, our approach has been to select reasonable values, that ensure changes in play selection within the short durations that make up a game ie. two 10 minute halves. In practice, we have found that these weight values are sufficient to ensure modified behavior within a half against a range of opponents of widely varying capabilities and strategies.

4.1 Evaluation

Our strategy system was used effectively during RoboCup 2002 against a wide range of opponents with vastly different strategies and capabilities. Although effective, the nature of robot competitions prevent them from being a systematic evaluation in a controlled, scientific, or statistically significant way. Therefore, we have constructed a number of simplified scenarios to evaluate our play system. These scenarios first compare whether multiple plays are actually necessary, and also examine the usefulness of playbook adaptation.

The scenarios compare the effectiveness of four different offensive plays paired against three different defensive behaviors. To simplify evaluation, only two offensive robots were used against one defensive robot. The robots start in the usual "kick off" position in the center of the field. For each scenario 750 trials were performed in our UberSim SSL simulator [2]. A trial was considered a success if the offense scored a goal within a 20 second time limit, and a failure otherwise. Table 5 lists the specifics of the offensive plays and defensive behaviors. The first two defensive behaviors, and first three offensive plays, formed the core of the playbook used for RoboCup 2002.

Table 6 shows the play comparison results. Each trial is independent, hence the maximum likelihood estimate of the play's success probability is the ratio of successes to trials. Note that there is no static strategy that is optimal. The best strategy depends upon the defensive behavior even in this simplified scenario. In fact, each of the offensive plays is actually the optimal response for some distribution over defense behaviors. These differences are statistically significant with 95% confidence using binomial difference tests.

Offensive Plays

Name	Attacker 1	Attacker 2
Shoot 1	`shoot Aim`	`position_for_rebound`
Shoot 2	`shoot NoAim`	`position_for_rebound`
Screen 1	`shoot Aim`	`mark 0 from_shot`
Screen 2	`shoot NoAim`	`mark 0 from_shot`

Defensive Behaviors

`block`	Positions to block incoming shots
`active_def`	Actively tries to steal ball
`brick`	Defender does not move

Table 5: List of offensive and defensive behaviors tested.

Play	`block`	`active_def`	`brick`
Shoot 1	**72.3%**	49.7%	**99.5%**
Shoot 2	66.7%	57.3%	43.1%
Screen 1	40.8%	59.0%	92.4%
Screen 2	49.2%	**66.0%**	72.0%

Table 6: Play comparison results. For each scenario, the percentage of successes for the 750 trials is shown. The boldfaced number corresponds to the play with the highest percentage of success for each defensive behavior.

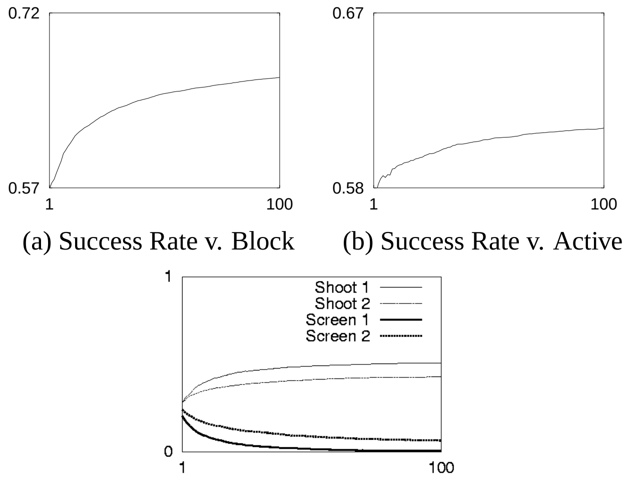
Our results support the notion that play-based strategies are capable of exhibiting many different behaviors with varying degrees of effectiveness. For instance, against a "conservative" blocking defense, the play where a robot takes the time to align itself for a good shot performs the best. On the other hand, against more "aggressive" defenses, the above play performs poorly in comparison. Instead, the play where the robots take less time to aim while the assisting robot attempts to set a screen for the shooter, is more successful.

To explore the effectiveness of playbook adaptation we use an offensive playbook for two robots with all four offensive plays described above against a fixed defender running either `block` or `active_def`. We initially used the algorithm above, but discovered an imperfection in the approach. Due to the strength of the reinforcement for a completed play, it is possible for a moderately successful but non-dominant play to quickly dominate, and remain dominant, in weight. This phenomenon did not occur in competition due to the larger disparity in plays against a given opponent and lower success probabilities. The problem is that there is no normalization in the weight adjustment for plays that have a higher selection probability, which are updated more often. Therefore, we included a normalization factor in the weight updates. Specifically, we used the following rule,

$$W(\mathbf{w}, p_i, o) = \begin{cases} w_{p_i} 2/\Pr(p_i) & \text{if } o = \texttt{Succeeded} \\ w_{p_i} \Pr(p_i)/2 & \text{if } o = \texttt{Failed} \end{cases},$$

where $\Pr(p_i)$ is the probability assigned to p_i according to $\mathbf{w}$.

To evaluate the performance, we compare the expected success rate (ESR) of using this adaptation rule against a fixed defensive behavior. We used the results in Table 6 to simulate the outcomes of the various play combinations. All the weights are initialized to 1. Figure 2(a) and (b) show the ESR for play adaptation over 100 trials, which is comparable to a length of a competition (approximately 20 minutes). The



(a) Success Rate v. Block (b) Success Rate v. Active

(c) Expected Play Probabilities v. Block

Figure 2: (a), (b) show ESR against block and active_def, (c) shows expected play success probabilities against block. These results have all been averaged over 50000 runs of 100 trials.

lower bound on the y-axis corresponds to the ESR of randomly selecting plays and the upper bound corresponds to the ESR of the playbook's best play for the particular defense. Figure 2(c) shows the probabilities of selecting each play over time when running the adaptation algorithm. Clearly, the algorithm very quickly favors the more successful plays.

These results, combined with the RoboCup performances, demonstrate that adaptation can be a powerful tool for identifying successful plays against unknown opponents. Note the contrast here between the use of adaptation to more common machine learning approaches. We are not interested in convergence to an optimal control policy. Rather, given the short time limit of a game, we desire adaptation that achieves good results quickly enough to impact the game. Hence a fast, but non-optimal response is desired over a more optimal but longer acting approach.

Finally, it is worth noting that play adaptation has a different role to tradition machine learning approaches. With the formulation described here, play adaptation does not allow for a team to perform better than the capabilities inherent in the underlying single robot skills (tactics in this case). Rather, it provides a team with an adaptation mechanism that ensures that its performance will be as good as possible, within the limits of the available tactics and the plays encoded in its playbook.

5 Conclusion

In conclusion, we have introduced a novel team strategy engine based on the concept of a play as a team plan, which can be easily defined by a play language. Multiple, distinct plays can be collected into a playbook where mechanisms for adapting play selection can enable the system to improve the team response to an opponent without prior knowledge of the opponent's strategy. The system was fully implemented for our CMDragons robot soccer system and tested at RoboCup 2002, and in the controlled experiments reported here.

References

[1] Andreas Birk, Silvia Coradeschi, and Satoshi Tadokoro, editors. *RoboCup 2001: Robot Soccer World Cup V*. Springer Verlag, Berlin, 2002.

[2] Brett Browning and Erick Tryzelaar. Ubersim: A multi-robot simulator for robot soccer. In *Proceedings of AAMAS*, 2003.

[3] James Bruce, Michael Bolwing, Brett Browning, and Manuela Veloso. Multi-robot team response to a multi-robot opponent team. In *ICRA Workshop on Multi-Robot Systems*, 2002.

[4] James Bruce and Manuela Veloso. Real-time randomized path planning for robot navigation. In *Proceedings of IROS-2002*, pages 2383–2388, Switzerland, October 2002.

[5] S.S. Intille and A.F. Bobick. A framework for recognizing multi-agent action from visual evidence. In *AAAI-99*, pages 518–525. AAAI Press, 1999.

[6] Itsuki Noda, Shóji Suzuki, Hitoshi Matsubara, Minoru Asada, and Hiroaki Kitano. RoboCup-97: The first robot world cup soccer games and conferences. *AI Magazine*, 19(3):49–59, Fall 1998.

[7] Patrick Riley and Manuela Veloso. Planning for distributed execution through use of probabilistic opponent models. In *ICAPS-02, Best Paper Award*, Toulouse, France, April 2002.

.

Goal-Converging Behavior Networks and Self-Solving Planning Domains

Bernhard Nebel
Institut für Informatik
Albert-Ludwigs-Universität Freiburg
nebel@informatik.uni-freiburg.de

Yuliya Babovich
University of Texas
yuliya@cs.utexas.edu

Abstract

Agents operating in the real world have to deal with a constantly changing and only partially predictable environment and are nevertheless expected to choose reasonable actions quickly. One way to address this problem is to use behavior networks as proposed by Maes, which support real-time decision making. Robotic soccer appears to be one domain where behavior networks have been proven to be particularly successful. In this paper, we analyze the reason for the success by identifying conditions that make behavior networks *goal converging*, i.e., allow them to reach the goals regardless of which particular action selection scheme is used. In terms of STRIPS domains one could talk of *self-solving planning domains*. We finally show that the behavior networks used for different robotic soccer teams have this property.

1 Introduction

Agents operating in the real world have to deal with a constantly changing and only partially predictable environment; and the expectation is that the agents can figure out the best suitable actions in real-time. The *behavior network* approach [Maes, 1990] addresses this problem through activation spreading inside a network of competence modules. This approach is intended to address, as Maes [1990] states, the problems of "brittleness, inflexibility, and slow response" of classical planning approaches on one hand, and the problem of "the lack of explicit goals" in reactive approaches on the other hand. It proved to be useful and became popular during the last decade. For instance, it has been used in the implementation of an intelligent e-mail agent [Zhang *et al.*, 1998] and as the underlying mechanism for generating behavior of autonomous characters in interactive story systems [Rhodes, 1996]. Most notably, the approach has been employed in the simulated robotic soccer team *magmaFreiburg* [Dorer, 2000a] and in the real robotic soccer (F2000 league) team *CS Freiburg* [Weigel *et al.*, 2001; 2002]. In both cases, the teams were highly successful. The simulation team *magmaFreiburg* was runner-up in 1999 and *CS Freiburg* won the *RoboCup* world championship in 2000 and 2001. Although there is a wide range of components, both hard- and software, that contribute to such a success, the behavior networks, as reported by Weigel *et al.* [2001; 2002] and Dorer [2000a], played a significant role.

It must be said, however, that the particular action selection mechanism employed in the robotic soccer teams differs significantly from Maes' [1990] original proposal. The so-called *extended behavior networks* [Dorer, 1999], which are used in the robotic soccer domain, can deal with continuous propositions, use a technique called *goal-tracking* in order to address some of Tyrrell's [1994] criticisms concerning Maes' [1990] proposal, and employ a goal management mechanism that allows for changing goals. In fact, with all the extensions, the behavior networks have now the flavor of decision-theoretic planning, without implementing this framework, though. Furthermore, as shown in a number of experiments [Dorer, 1999], these changes lead to a significantly higher number of scored goals.

Although Maes' behavior networks and variations have been analyzed from several perspectives, there are nevertheless many issues that have not been resolved. For example, it is not clear under which conditions we can be sure that a behavior network converges to its goal, i.e., generates an action sequence that eventually satisfies the goal. Dorer [2000b] describes some experiments where he used the original behavior networks by Maes [1990] in order to solve *blocks-world planning* problems. As it turns out, for some five-block problems, the behavior network goes into an infinite loop and does not come up with a solution, regardless of the parameter setting. Clearly, such a performance would be unacceptable in a soccer context. Just imagine a soccer player who dribbles the ball in an endless circle. However, this does not happen in this domain. One could explain this difference by the fact that the blocks world is an artificial domain with a puzzle-like character while soccer has a real-world character much more suited for behavior networks. However, it would be, of course, more interesting to find some formal conditions that explain why behavior networks work so well for robotic soccer.

More generally, we are interested to find a condition that guarantees that the behavior network will generate a successful sequence of actions provided there exists one and no exogenous events intervene. Furthermore, we want this guarantee regardless of which particular action selection scheme and parameter setting is employed. Behavior networks with this property will be called **goal converging**. If we view the behavior network as a STRIPS planning domain specification, then the corresponding domain specification could be termed *self-solving*, since all sequences of executable actions lead to the goal.[1]

If a behavior network is goal converging, then we know that it will always act goal-oriented and parameter tuning is only necessary to generate better, shorter action sequences. Of course, it is also clear that goal convergence will require severe restrictions on the structure of behavior networks. However, as we show in this paper, there exists a non-trivial restriction on the topology of the behavior network that guarantees that the network is goal converging. In addition, all the networks that have been designed for robotic soccer are of this type (or are very close to this form), which explains to some degree why the approach works so well for robotic soccer.

The rest of the paper is structured as follows. In the next section we sketch the behavior network approach. In Section 3, we identify two conditions for a behavior network being goal converging. Based on that, we analyze in Section 4 the networks that have been used in the Freiburg RoboCup teams and show that they satisfy one of the conditions identified. Finally, in Section 5, we conclude and give an outlook.

2 Behavior Networks

In the following, we describe the behavior network formalism. Since we do not need the full details for our purposes, the description will be sketchy at some points.

2.1 Specifying Behavior Networks

Let $\mathcal{P}$ be a set of propositional atoms. A **state** is a truth assignment to all atoms in $\mathcal{P}$ (often also represented as the set of true atoms). For extended behavior networks [Dorer, 1999], the state is an assignment of fuzzy values. **Behavior networks** are tuples $(\mathcal{P}, \mathcal{G}, \mathcal{M}, \Pi)$, where

- $\mathcal{G} \subseteq \mathcal{P}$ is the **goal specification**;
- $\mathcal{M}$ is a finite set of **competence modules** or **actions**, where $m \in \mathcal{M}$ is a tuple $\langle \textit{pre}, \textit{eff}^+, \textit{eff}^-, \textit{beh} \rangle$ with $\textit{pre} \subseteq \mathcal{P}$ denoting the **preconditions**,[2] $\textit{eff}^+, \textit{eff}^- \subseteq \mathcal{P}$ denoting the positive and negative effects, respectively, with $\textit{eff}^+ \cap \textit{eff}^- = \emptyset$ and *beh* being the name of an executable **behavior**, which is started once the module is selected for execution. If we want to refer to one of the components of a competence module m we use the notation $\textit{pre}(m), \textit{eff}^+(m)$, etc.
- Π is a set of **global parameters** used to control the action selection process, among them the threshold for the activation θ. There are more parameters, but we do not need them for our purposes and ignore them for this reason.

Depending on the type of behavior networks, some variations are possible. For example, in Dorer's [1999] extended behavior networks, the goals can have an importance measure and an additional relevance condition. Further, effects have an expectation value describing how likely it is that the effect proposition becomes true after executing the competence module. These details will not be important for us, though.

2.2 Activation Spreading

Competence modules are connected in a network so that they can send and receive activation energy. A *positive effect link* connects a positive effect p of a competence module to the precondition p of another competence module. A *negative effect link* connects a negative effect p of one competence module to the precondition p of another competence module.[3] An example of a small behavior network is given in Figure 1.

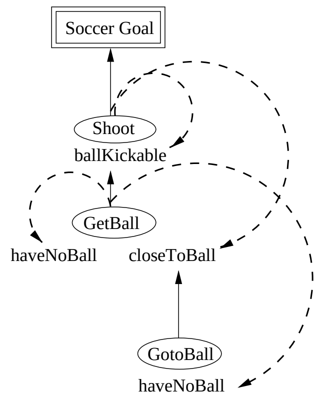


Figure 1: Example of a behavior network: Solid arrows denote positive effect links and dashed arrows denote negative effect links.

In this example, the competence module *GotoBall* has the precondition *haveNoBall* and the effect *closeToBall* enabling the competence module *GetBall*. This, in turn, has the negative effect of deleting *haveNoBall* and the positive effect of making *ballKickable* true. The latter enables the *Shoot* module, which then (hopefully) leads to scoring a goal, the ultimate goal of this behavior network.

Unsatisfied goals send some activation energy to competence modules that could make the goals true and, in turn, each activated module sends some of its activation through its unsatisfied preconditions to modules which can make the precondition true. In the original version of behavior networks, there is also a "forward spreading" of activation energy. This means that activation energy flows from propositions true in a

[1]This condition corresponds to what is called the *all-policies-proper* condition in the MDP community. However, in this context one usually assumes the condition and does not try to identify criteria which guarantee the condition.

[2]Note that we allow only for positive goals and preconditions. This, however, does not restrict the expressivity since (for STRIPS-like planning) this is equivalent to formalisms with negative preconditions and goals under various formal notions of expressive equivalence [Bäckström, 1995; Nebel, 2000].

[3]Although negative self-links are usually not considered, we will draw them in depictions of behavior networks in order to describe the actions completely.

situation towards competence modules that have these propositions as preconditions, and from executable competence modules to competence modules which have unsatisfied preconditions identical to the effects of the executable modules. However, this forward spreading of activation does not seem to increase the quality of the action selection [Dorer, 1999; Goetz and Walters, 1997] and for this reason this kind of activation is not present in Dorer's [1999] extended behavior networks. While positive effect links are used for spreading activation, negative links are used to inhibit the activation of other modules. Modules that have the negative effect $p \in \mathit{eff}^-$ are inhibited by modules that have p as a satisfied precondition.

2.3 Action Selection

Action selection is done in a cycle containing four steps [Maes, 1990; Dorer, 1999]:

1. The current activation of each module is calculated using the methods described above, i.e., each modules receives some activation and inhibition from modules connected to it.
2. Activation and executability of a module are combined by a non-decreasing function into the utility of a module, whereby non-executable competence modules always get the value zero.
3. The module with the highest utility value is chosen,[4] provided it passes a certain threshold θ (one of the global parameters). The action associated with the competence module is then executed.
4. If none of the modules reached the activation threshold, the threshold is reduced by a certain percentage (another global parameter) and the cycle is started again with the currently computed activation values for each module.

Since we usually want an agent to execute a sequence of actions leading to the goal, the above cycle will be called infinitely or until the agent has reached the goal.

From the description above it follows that there are only a few things one can be sure of when using a behavior network for action selection. First of all, only executable actions are chosen. Second, if an action selection scheme is employed that does not use forward activation spreading, for instance Dorer's [1999] scheme, then it follows that if an action is chosen, it "contributes" to one of the goals, since the competence module can receive activation only from the goal through a chain of unsatisfied preconditions.

2.4 Ideal Abstract Behavior Networks

If we want to guarantee properties of a network under different action selection schemes and parameter settings, we have to make a number of simplifying assumptions. We will assume that the state is always correctly observable (with Boolean state variables), that the competence modules describe all relevant effects correctly, that the execution of the behavior of a competence module is always successful, and that no exogenous event will intervene. Based on these assumptions, we define an abstract version of behavior networks, which from a formal point of view are identical to STRIPS domain descriptions.

An **ideal, abstract behavior network** is a tuple $\mathbf{B} = (\mathcal{P}, \mathcal{G}, \mathcal{M})$, where $\mathcal{P}, \mathcal{G}$ and $\mathcal{M}$ are defined as in Section 2.1. In the state $S \subseteq \mathcal{P}$, the network can choose any competence module m for execution such that the preconditions $\mathit{pre}(m)$ are satisfied in S, i.e., $\mathit{pre}(m) \subseteq S$, and not all positive effects are satisfied, i.e., $\mathit{eff}^+(m) - S \neq \emptyset$. When m is executed in state S, the resulting state $\mathit{Result}(S, m)$ is given by

$$\mathit{Result}(S, m) = S - \mathit{eff}^-(m) \cup \mathit{eff}^+(m).$$

We say that the network $\mathbf{B}$ can **generate** a (finite or infinite) sequence of actions $m_1, m_2, \ldots, m_i, \ldots$ in a state S_1 if

$$S_{i+1} = \mathit{Result}(S_i, m_i).$$

We say $\mathbf{B}$ can **reach the goals** $\mathcal{G}$ from a state S if it can generate a finite sequence of actions in S such that the last state S_n satisfies the goals, i.e., $S_n \supseteq \mathcal{G}$.

3 Goal-Converging Behavior Networks

If we want to guarantee that a behavior network is successful regardless of the action selection scheme and parameter setting,[5] we have to consider all action sequences the network can generate. Although this appears to be a fairly strong requirement, there are indeed realistic networks for which we can show that they are always successful—if the goal is reachable at all.

3.1 Terminating and Dead-End Free Networks

We call a behavior network **terminating** if for all states and under all possibilities to choose actions, it is impossible to generate infinite action sequences—provided the goal was reachable initially.[6] Figure 2 gives a simple example of a non-terminating network.

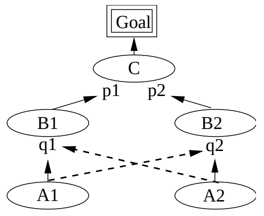


Figure 2: A *non-terminating* behavior network

Provided that $p1, p2, q1, q2$ and the *Goal* are false initially, then it is possible that the sequence $A1, A2, A1, A2, \ldots$ is chosen. Hence, the network is not terminating. Note that there is a successful sequence consisting of

[4]Ties are broken randomly.

[5]The only restriction is that we never consider actions such that all their positive effects are already satisfied (see Section 2.4).

[6]If the goal is unreachable, we do not care about the behavior of the network.

$A1, B1, A2, B2, C$. However, the action selection mechanism might not necessarily find it. An example for a terminating network is the one in Figure 1, as is easy to verify.

We say that a network is in a **blocked state** when no action is executable and the goal is not satisfied. Such a blocked state may occur because there was no way to reach the goal in the first place. However, it may be possible that the goal was reachable in the beginning. We call a network **dead-end free** if it never leads to a blocked state when it is possible to reach the goal. Consider, for example, the network in Figure 3. This network contains a dead end. Provided one starts with $p1, p2, q2$ and *Goal* as false and $q1$ as true, the execution of $A2, B2$ leads to a blocked state. However, obviously, the sequence $B1, A2, B2, C$ would have led to the goal. In other words, this network is not dead-end free. An example of a dead-end free network is again the one in Figure 1. Although in this network one can make propositions false, this can only happen in the course of satisfying the goal and it will never prohibit reaching the goal.

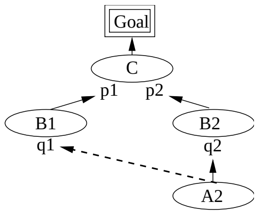


Figure 3: A behavior network with a *dead end*

Finally, we call a behavior network **goal converging** when it will generate a finite action sequence leading to the goal regardless of the action selection scheme and parameter setting, provided the goal is reachable at all. When viewing the behavior networks as specifications of STRIPS planning problems, we would talk of **self-solving** planning domains, because regardless of which order we would choose for the executable actions, one would always reach the goal—provided the goal was initially reachable at all.

Proposition 1 *A behavior network is* goal converging *if and only if it is* dead-end free *and* terminating.

Proof: The “only if” direction is obvious since networks with dead ends and networks which are non-terminating cannot be goal converging. There are possible states and action selections such that either a loop or a dead end, respectively, are chosen although there is the possibility of reaching the goal. For the “if” direction observe that a non-goal-converging network must either produce an infinite sequence or end up in a dead end although there is a action sequence leading to a goal state. ■

3.2 Monotone Networks

One particularly simple type of goal-converging networks are networks with only positive effects, which we will call **monotone networks**. Since a propositional atom can never be made false in a monotone network, one can reach any desired goal after any initial sequence of actions, provided the goal was initially reachable. This implies that it is impossible to run into a dead end. Since each action can be executed at most once, there is additionally an upper bound to the length of any action sequence generated by a monotone behavior network, implying that the network is also terminating.

Proposition 2 *Monotone behavior networks are goal converging.*

Monotone behavior networks are hardly interesting, because they almost never appear in practice.For our purposes, they are equivalent to STRIPS planning problems that have only positive preconditions and effects. While such planning problems appear to be trivial, it is well known that generating a shortest plan is still an NP-hard problem [Bylander, 1994]. Furthermore, such planning problems have become popular as the basis for computing heuristic estimates in action planning [Hoffmann and Nebel, 2001; Bonet and Geffner, 2001]. For our purposes, however, the restriction to purely positive effects is not possible.For instance, in our example network in Figure 1, the action *GetBall* destroys the *haveNoBall* condition.

3.3 Acyclic Networks with Restricted Negative Links

In order to specify a more interesting class of goal-convergent networks, let us view these networks from a slightly different angle. Let us consider directed graphs with two kinds of nodes, **action nodes** and **fact nodes** and two kinds of directed edges, **positive** and **negative** ones, such that

- there is a positive (precondition) edge from fact node p to action node a if p is a precondition of action a;
- there is a positive (effect) edge from action node a to fact node p if p is a *positive* effect of a;
- there is a negative (effect) edge from action node a to fact node p if p is a *negative* effect of a.

The resulting graph is called **action-fact graph**.[7] The **normalized action-fact graph** is the directed graph where the direction of the negative edges has been reversed. The interesting point is that acyclicity of the normalized action-fact graph implies that the behavior network is terminating.

Theorem 3 *A behavior network which corresponds to an acyclic normalized action-fact graph is terminating.*

Proof: In order to show that a behavior network satisfying the condition of the theorem is terminating, we assign as a first step values to the atoms in the action-fact graph. For each atom p the value of p should be 1 plus the sum of values of the fact nodes that are incident via a negative edge to an action having p as a positive effect. Since the normalized action-fact graph is acyclic, this value assignment is well-defined.[8]

With this value assignment to atoms, each action application will strictly increase the overall value of the state (as the

[7]Such graphs correspond to what has been called *bi-level planning graph* [Long and Fox, 1999] or *connectivity graph* [Hoffmann and Nebel, 2001] in the planning literature.

[8]In fact, as is obvious from this argument, it suffices when the sub-graph consisting of effect edges only is acyclic.

sum over the values of all true propositions), because an action is only executed when one of its positive effects is not true. This implies, however, that it is impossible to generate infinite action sequences. ■

While it was easy to find a condition for termination, it appears to be much more difficult to find a criterion that guarantees that the network is dead-end free. Let us consider even further restricted action-fact graphs. If the sub-graph formed from the positive links is acyclic and if for all negative edges from action a to fact p there exists a positive path from p to a, then we call the graph **acyclic, negative-feedback action-fact graphs**. This is obviously a special-case of an *acyclic normalized action-fact graph*. However, it is still not a criterion for guaranteeing the absence of dead ends. In fact, planning is still non-trivial as the plan existence problem is still NP-hard.

3.4 Modular Action-Fact Graphs

One way to guarantee that there are no dead ends is to make sure that it is always possible to make falsifiable propositions true without affecting other propositions, which has to be guaranteed independently of the initial state[Hoffmann, 2002]. While this condition is often true in classical planning tasks, it seems very unlikely that we can guarantee this in our case. Hoffmann [2002] gives a number of other sufficient conditions, but none appears to be applicable here. For this reason, we will look into an alternative condition. We will try to make sure that any proposition that can be falsified needs never to be used again after it has been falsified. For example, this condition is satisfied in Figure 1. One way to guarantee this is to require the following *modularity* condition. For all atoms q that can be falsified by an action a in an acyclic, negative-feedback action-fact graph, each positive path from q to a goal atom must go through an action a' such that $\mathit{eff}^+(a) \supseteq \mathit{eff}^+(a') \neq \emptyset$. This condition is, for example, satisfied by the action-fact graph in Figure 4 and the action-fact graph derivable from the network in Figure 1. We call acyclic, negative-feedback action-fact graphs satisfying this condition **modular action-fact graphs**.

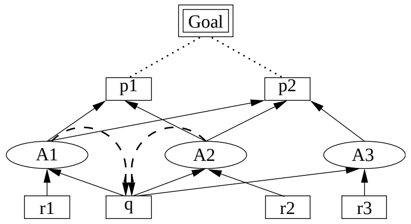


Figure 4: An action-fact graph satisfying the *modularity* condition

Theorem 4 *Modular action-fact graphs are goal converging.*

Proof: Termination follows from Theorem 3. The proof that the action-fact graphs are dead-end free is by induction on the number of negative links. For $k = 0$ negative links, the claim follows from Proposition 2. Assume now that the claim is true for modular action-fact graphs with k or fewer negative links. Consider a graph with $k+1$ negative links. Now choose one action node a that is the source of a negative link and which has no positive path to any other action node with such a property. Because of the acyclicity of the graph formed from positive links, such a node must exist. Assume that q is amongst the negative effects of a and that the positive effects are $p_1, \ldots, p_k$. If we remove the negative link from a to q, we can apply the induction hypothesis for k negative links and know that the graph is dead-end free.

Assume now for contradiction that the original network is not dead-end free. This must be connected with the possibility of falsifying q by a. However, once all the positive effects of a have been made true by executing a, the truth value of q is not of any concern since all positive paths from q to a goal go through a and actions with a subset of $\mathit{eff}^+(a)$ as their positive effects. Hence, the negative link from a to q cannot create a dead end, which completes the induction step. ■

4 RoboCup Behavior Network

As mentioned in the Introduction, the analysis of behavior networks was motivated by the observation that the behavior networks of the *magmaFreiburg* and *CS Freiburg* robotic soccer players work so robustly. When one now analyzes the networks with the tools developed in this paper, it turns out that they indeed satisfy the condition of being *modular*—modulo some qualifications. Before we talk about qualifications, we should, however, have a look at some real behavior networks. In Figure 5 the main part of the *CS Freiburg* [Müller, 2000] behavior network is displayed as an action-fact graph. Obviously, the few negative links satisfy the *modularity* condition. However, one may wonder, why there are no negative links from the actions having *HaveBall* as a precondition to *HaveBall*? Although these negative links should have been there in order to describe the action effects correctly, their absence is not problematic, since we assumed that all actions are successful—and the positive effect of all the actions is the ultimate goal. In any case, when adding the negative effects, we still would have a *modular* action-fact graph.[9]

A similar comment applies to the missing positive links back to *NegHaveBall*. Again, it is not interesting because we achieve the goal anyway. Furthermore, we can ignore these positive links without losing anything, i.e., they never help us to achieve the goal.

Often it is necessary to take more than one goal into account. The extended behavior network may contain multiple goals which can be selected based on the current situation. So, for example, a *CS Freiburg* player either tries to score a goal (if it fills the role of an *active player*) or it has the overall goal to *cooperate*. In the latter case, we would have to consider a different network, which also satisfies the structural condition of being *modular*, though. In the case of the *magmaFreiburg* players, things are even more complicated because it is possible to pursue more than one goal at once. If we break the networks down to one goal at a time, however, the resulting networks are again modular.

[9]Indeed, the *magmaFreiburg* networks contain these negative effects.

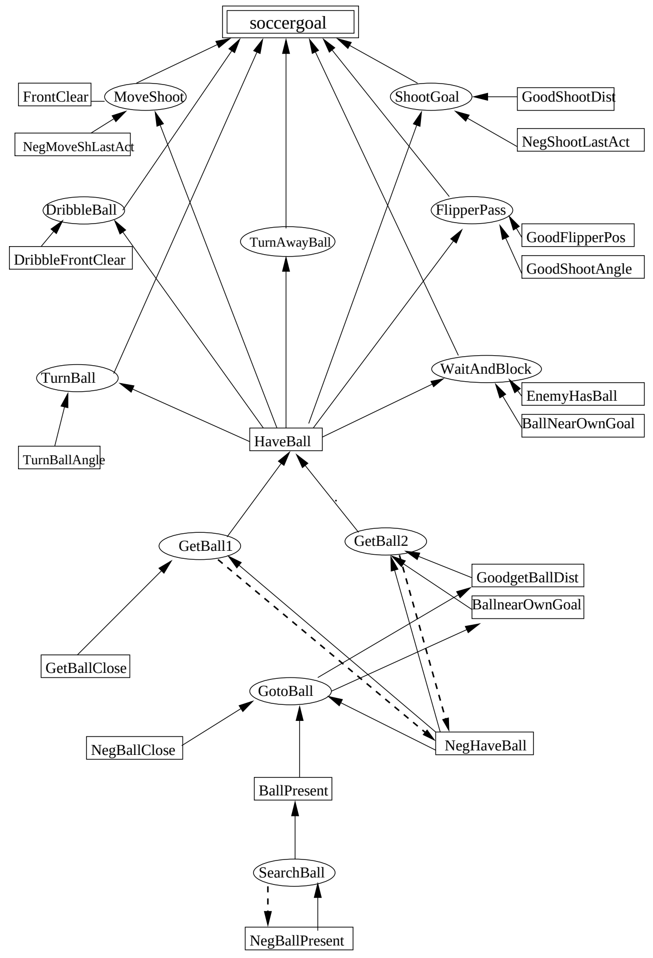


Figure 5: Part of the Action-Fact Graph of the *CS Freiburg* behavior network [Müller, 2000]

Finally, it should be noted that there are levels in the decision making that influence the behavior networks, e.g., the role assignment and placement of players on the field [Weigel *et al.*, 2001], which are, however, not taken into account when analyzing the network.

Summarizing, if we assume that no exogenous actions intervene and if there occurs no change in the goals (in particular there is no influence from the strategic component), then all the behavior networks of the *CS Freiburg* [Müller, 2000] and the *magmaFreiburg* [Dorer, 2000b] players satisfy the modularity condition and are therefore goal converging, which goes somewhere in explaining why they have been successful. At least, when players are alone on the field, they will eventually score. Although this is a rather weak guarantee, it is much better than the statement that the player might score a goal only when the parameters of the network are well adjusted.

Of course, all this seems to imply that the domain as modelled in the described RoboCup teams has a quite simple structure. However, thinking a while about the problem, one will come to the conlusion that even in the face of more complex modelling and decision making by, e.g., integrating opponent modelling and adversary planning, we nevertheless would like to guarantee the conditions mentioned above. However, it may be the case that it is not possible to verify the conditions using simple syntactic tests any longer.

5 Conclusions and Outlook

We have identified a structural property of behavior networks, called *modularity*, that guarantees that the networks will reach their goals in a static environment under all circumstances—if the goals are reachable at all. Interestingly, there exists a significant application of behavior networks where this restriction is met, namely, the networks of the Freiburg simulation and real robot (F2000) soccer players.

Having shown that a network has this property means that we never have to fear that the network leads to infinite action sequences or blocked states. In addition, it means that tuning network parameters [Maes, 1992] will not modify the principal property of reaching the goal, but only the efficiency.

In the future, we will pursue three directions of research. First of all, there is the question whether there exist other relevant restrictions on network structures that lead to goal convergence. Second, in most cases, it is enough if the network is goal converging for a subset of all possible states. Now, the interesting question is in how far this would result in a more liberal condition for goal convergence. Third, we will analyze the feasibility of testing the property of goal convergence on a semantic level. In this context, it will probably be helpful to take the syntactic restrictions identified in this paper into account, because it is probably prohibitive to inspect the entire state space.

Acknowledgments

We acknowledge the comments and suggestions by Jörg Hoffmann, Wolfram Burgard, and Malte Helmert on an earlier version of this paper.

The first author has been partially supported by the University of Newcastle, NSW, Australia, during his sabbatical. The second author has been partially supported by the DAAD as part of the International Quality Network on Spatial Cognition during a visit at the University of Freiburg.

References

[Bäckström, 1995] Christer Bäckström. Expressive equivalence of planning formalisms. *Artificial Intelligence*, 76(1–2):17–34, 1995.

[Bonet and Geffner, 2001] Blai Bonet and Héctor Geffner. Planning as heuristic search. *Artificial Intelligence*, 129(1-2):5–33, 2001.

[Bylander, 1994] Tom Bylander. The computational complexity of propositional STRIPS planning. *Artificial Intelligence*, 69(1–2):165–204, 1994.

[Dorer, 1999] Klaus Dorer. Behavior networks for continuous domains using situation-dependent motivations. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI-99)*, pages 1233–1238, Stockholm, Sweden, August 1999. Morgan Kaufmann.

[Dorer, 2000a] Klaus Dorer. The magmaFreiburg soccer team. In M. Veloso, E. Pagello, and H. Kitano, editors, *RoboCup-99: Robot Soccer World Cup III*, pages 600–603. Springer-Verlag, Berlin, Heidelberg, New York, 2000.

[Dorer, 2000b] Klaus Dorer. *Motivation, Handlungskontrolle und Zielmanagement in autonomen Agenten*. PhD thesis, Albert-Ludwigs-Universität, Freiburg, Germany, 2000. Published on FreiDok server under http://www.freidok.uni-freiburg.de/volltexte/57.

[Goetz and Walters, 1997] Philip Goetz and Deborah Walters. The dynamics of recurrent behavior networks. *Adaptive Behavior*, 6(2):247–283, 1997.

[Hoffmann and Nebel, 2001] Jörg Hoffmann and Bernhard Nebel. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14:253–302, 2001.

[Hoffmann, 2002] Jörg Hoffmann. Local search topology in planning benchmarks: A theoretical analysis,. In *Proceedings of the 6th International Conference on Artificial Intelligence Planning Systems (AIPS-02)*. AAAI Press, Menlo Park, 2002.

[Long and Fox, 1999] Derek Long and Maria Fox. Efficient implementation of the plan graph in STAN. *Journal of Artificial Intelligence Research*, 10:87–115, 1999.

[Maes, 1990] Pattie Maes. Situated agents can have goals. In Pattie Maes, editor, *Designing Autonomous Agents: Theory and Practice from Biology to Engineering and Back*, pages 49–70. MIT Press, Cambridge, MA, 1990.

[Maes, 1992] Pattie Maes. Learning behavior networks from experience. In *Proceedings of the First European Conference on Artificial Life*, pages 48–57, 1992.

[Müller, 2000] Klaus Müller. Roboterfußball: Multiagentensystem CS Freiburg. Diplomarbeit, Albert-Ludwigs-Universität, Freiburg, Germany, 2000.

[Nebel, 2000] Bernhard Nebel. On the compilability and expressive power of propositional planning formalisms. *Journal of Artificial Intelligence Research*, 12:271–315, 2000.

[Rhodes, 1996] Bradley Rhodes. PHISH-nets: Planning heuristically in situated hybrid networks. Master's thesis, MIT, Cambridge, MA, 1996.

[Tyrrell, 1994] Toby Tyrrell. An evaluation of Maes' bottom-up mechanism for behavior selection. *Adaptive Behavior*, 2(4):307–348, 1994.

[Weigel *et al.*, 2001] Thilo Weigel, Willi Auerbach, Markus Dietl, Burkhard Dümler, Jens-Steffen Gutmann, Kornel Marko, Klaus Müller, Bernhard Nebel, Boris Szerbakowski, and Maximilian Thiel. CS Freiburg: Doing the right thing in a group. In P. Stone, T. Balch, and G. Kraetzschmar, editors, *RoboCup-2000: Robot Soccer World Cup IV*, pages 52–63. Springer-Verlag, Berlin, Heidelberg, New York, 2001.

[Weigel *et al.*, 2002] Thilo Weigel, Alexander Kleiner, Florian Diesch, Markus Dietl, Jens-Steffen Gutmann, Bernhard Nebel, Patrick Stiegeler, and Boris Szerbakowski. Cs freiburg 2001. In A. Birk, S. Coradeschi, and S. Tadokoro, editors, *RoboCup-2001: Robot Soccer World Cup V*. Springer-Verlag, Berlin, Heidelberg, New York, 2002.

[Zhang *et al.*, 1998] Zhaohua Zhang, Stan Franklin, Brent Olde, Yun Wan, and Art Graesser. Natural language sensing for autonomous agents. In *Proceedings of the IEEE Joint Symposia on Intelligence and Systems*, pages 374–381, Rockville, Maryland, 1998.

.

An Approach to Predicting the Location of Moving Objects During On-Road Navigation

Craig Schlenoff, Raj Madhavan, Stephen Balakirsky
Intelligent Systems Division
National Institute of Standards and Technology
100 Bureau Drive, Stop 8230
Gaithersburg, MD 20899
Phone: 301-975-3456 F: 301-990-9688
craig.schlenoff@nist.gov, raj.madhavan@nist.gov, stephen.balakirsky@nist.gov

Abstract

For an autonomous vehicle to navigate in real-time within a dynamic environment, it must be able to respond to moving objects. In particular, it must be able to predict, with appropriate levels of confidence, where those objects are expected to be at times in the future. It must then capture this information internally in its world model in a format amenable for planners that intend to use it.

In this paper, we provide an overview of a framework to address the challenges involved in predicting and representing the future location of moving objects. This framework uses a multi-representational approach to model information about moving objects, thus allowing for planners that require different forms of knowledge representation. We then describe a probabilistic, logic-based algorithm to predict the future location of vehicles in an on-road environment. Included in this discussion are the factors that affect the probabilities associated with various actions that the moving object may take.

1 Introduction and Related Research

For an autonomous vehicle to navigate in real-time within a dynamic environment, it must be able to respond to moving objects. In particular, it must be able to predict, with appropriate levels of confidence, where those objects are expected to be at times in the future. It must then capture this information internally in its world model in a format amenable for planners that intend to use it.

In this paper, we introduce a framework to address the challenges involved in predicting and representing the future location of moving objects. This framework uses a multi-representational approach to model information about moving objects, thus allowing for planners that require different forms of knowledge representation. In addition, this framework accounts for different factors that would influence the future location of moving objects, including the environment it is in, *a priori* maps, mobility characteristics, the object's intention and indicators, environmental conditions, etc.

This framework is applicable to both on-road and off-road driving. However, for the remainder of this paper, we will be focusing on the more interesting problem of on-road navigation where road networks provide a constrained environment in which to navigate, and as such, introduce a number of additional factors that could influence the probability of other moving objects behaving in certain ways. These factors and their corresponding influences are the focus of this paper.

We are not aware of any efforts in the literature that have addressed the development of a framework for combined moving objects representation and prediction. However, there have been efforts focusing on individual components of this framework that could be leveraged, specifically in moving object representation. Very limited work exists in the representation of moving objects. Firby [4] uses NaTs (navigation templates) as a symbolic representation of static and dynamic sensed obstacles to drive a robot's motors to respond quickly to moving objects. Gueting [5] extends database structures to allow for the representation of dynamic attributes (i.e., ones that change over time) and also extends the database's query language to allow for simplified querying of the values of dynamic attributes. Singhal [10] introduces the concept of dynamic occupancy grids which allow each cell to have a state vector which contains information such as a probabilistic estimate of the entity's identity, location, and characteristics (such as velocity, acceleration) along with global probability distribution functions.

In the literature, it is common to find methodologies that predict where an object will be in the next one or two sensor images. This form of predicting the future location of moving objects for a relatively small number of time steps into the future (short-term prediction) is useful in determining where the object will be in the next sensor image so as to perform object tracking. This has been a well-researched area in which approaches including Kalman filters [3] and Bayesian-based methods [11] have shown good results. In this paper, we concentrate on long-term prediction of sensor images, i.e., predicting the position of objects 10's or 100's of sensor images in the future. For autonomous navigation, planners plan over time horizons anywhere from milliseconds to tens of seconds for path planning and obstacle avoidance. As such, prediction algorithms need to be able to generate both short and long term predictions to accommodate the needs of planners.

In this paper, we introduce an approach to predicting the future location of moving objects with the following characteristics:

- The recipient of the predictions is the planner and are used for path planning and obstacle avoidance
- Predictions are made at longer time horizons, on the order of 10's or 100's of sensor images into the future
- Constraints on the environment are explicitly taken into account, such that only legal and possible actions are considered during prediction
- A logic-based approach is used to associate probabilities with various actions the moving object may take.

In Section 2, we provide an overview of the moving object framework. In Section 3, we describe the prediction algorithms and apply them to on-road driving. In Section 4, we discuss the influencing factors and constraints on motion that affect the probabilities associated with actions used in the prediction algorithms. In Section 5, we discuss the implications of applying this approach to an existing planner. In Section 6, we conclude the paper and discuss future work.

2 Moving Object Framework

The moving object framework provides a mechanism to apply appropriate prediction algorithms and representational approaches in order to fully capture the information needed to navigate in the presence of moving objects. This framework is shown in Figure 1.

We are assuming that the processed sensor data will be provided as input to the framework (as shown in the left-most box). Specifically, the information that we are assuming will be provided at pre-defined time intervals includes:

- The perceived dimensions of objects in the environment, along with associated uncertainty
- The location of objects in the environment
- The object's velocity and direction
- The color of the object

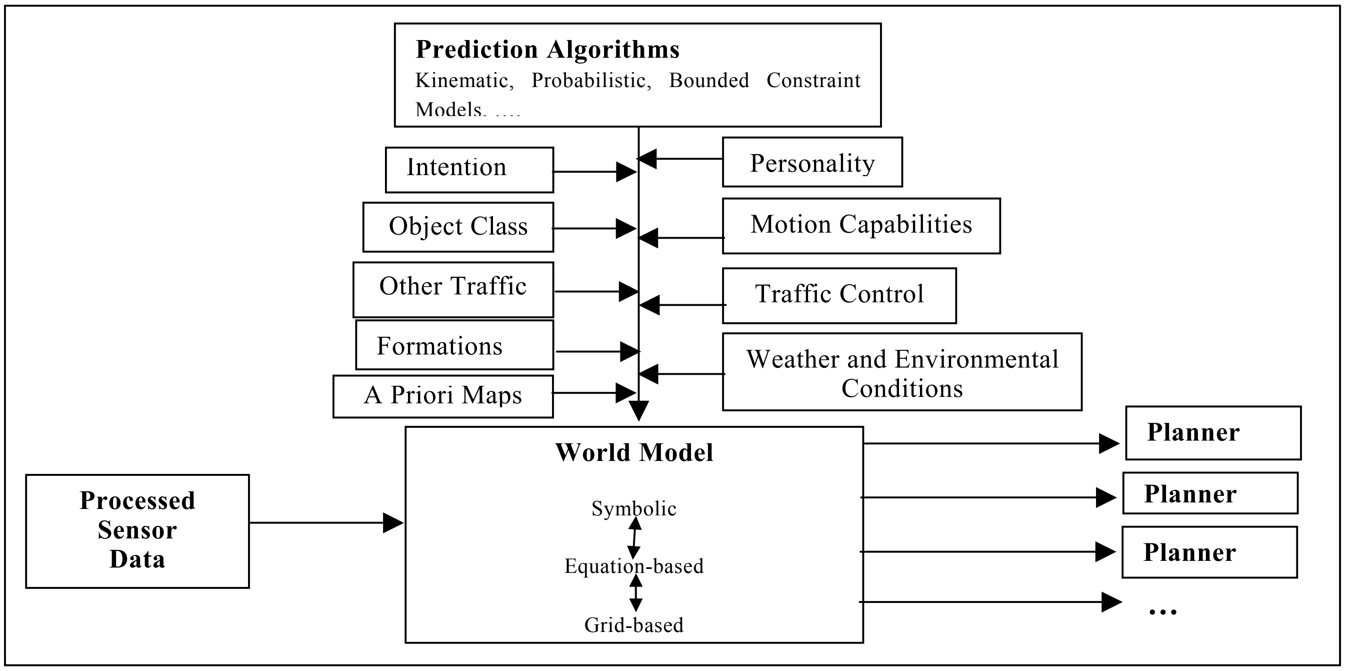


Figure 1: Moving Object Framework

We are also assuming knowledge about the environment in which the vehicle is navigating. This could take the form of *a priori* maps containing road networks and terrain characteristics, or could be dynamically generated based upon sensory input and processing.

In the framework, we attempt to classify the moving object based upon the processed sensor data. In the case of on-road driving, simply classifying the objects as vehicles (cars, motorcycles, trucks, buses, emergency vehicles), pedestrians, animals, or debris is enough for the purpose of motion prediction. We introduce a fairly simplistic object classification algorithm in [8] to provide the level of classification necessary to allow for informed moving object prediction.

Based upon the environment we are in, we employ different types of prediction algorithms. For off-road navigation, we have employed a bank of Kalman filters to predict the future location of the moving objects in the environment [6]. For on-road navigation, we are developing logic-based prediction algorithms that are intended to function in constrained environments. This logic-based approach is discussed in Section 3 of this paper.

Information about the moving object and its possible future locations are stored in the vehicle's world model in a multi-representational format. Information about the instances of object classes encountered in the environment (e.g., vehicle, animal, pedestrian, debris) is stored in a symbolic knowledge base with links to *a priori* detailed information about the corresponding object class. Based on the object class and the environment in which it is in, prediction algorithms, as discussed in the previous paragraph, are linked to the symbolic representations of the object. The results from these prediction algorithms are instantiated and represented in a time-based grid representation and provided to lower-level planners. A detailed discussion of these representation formalisms and their interactions can be found in [9].

One of the major advantages the proposed moving object framework provides is the ability to represent information about the moving object in many different, inter-related representations. It is expected that this moving object framework will provide information to planners that require fundamentally different kinds of underlying representations. For example, a planner that is planning for short time horizons (on the order of a few seconds) may require a grid-based representation describing occupancy probabilities of locations in space while a planner that plans at a longer time horizon may require a symbolic representation that describes characteristics of objects and equations governing their motion as opposed to locations in space. By using an interconnected multi-representational approach, we are able to provide information to the planner that is at a level of abstraction appropriate to its planning requirements. More information on planning in the presence of moving objects can be found in Section 5.

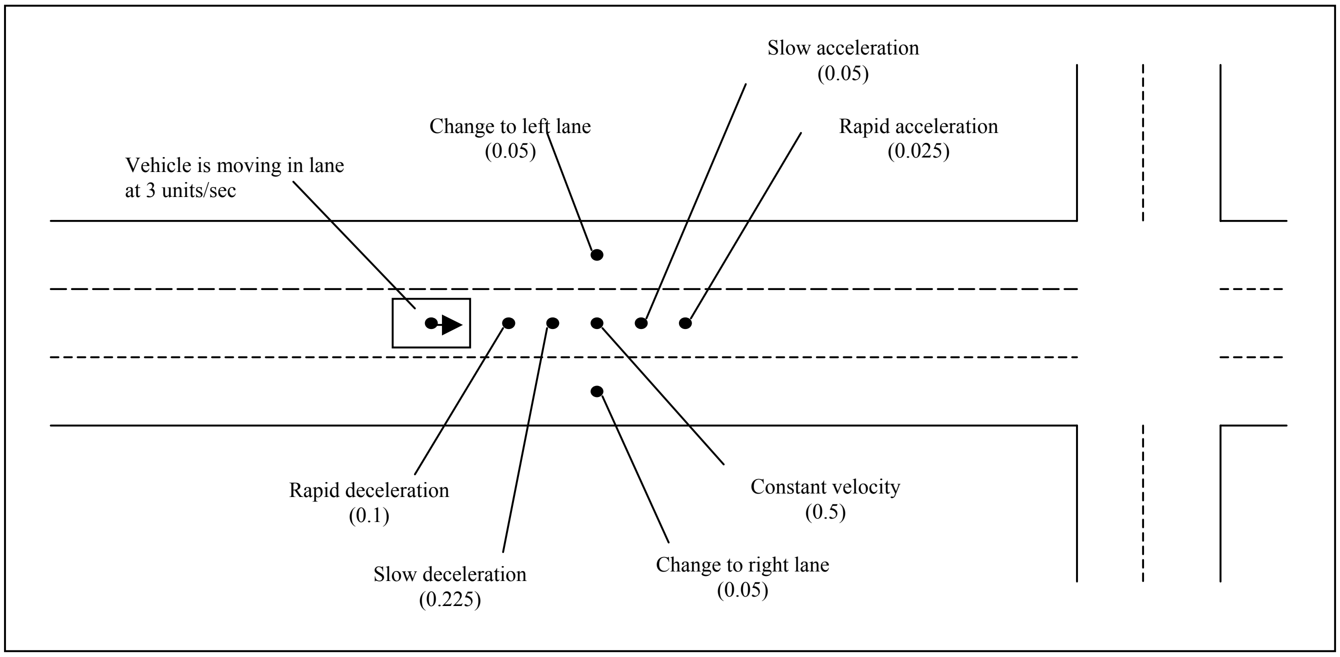


Figure 2: Driving Scenario

3 Logic-Based Motion Predictions in Constrained Environments

We are developing logic-based prediction algorithms for use in constrained environments. The purpose of these algorithms is to predict the probability that an object will occupy a given location in space at a given time by taking into account: 1) the constraints that are placed on the object's motion and 2) the influencing factors that would cause it to take a given action over another at specific times. These constraints and influencing factors are discussed in Section 4 of this paper.

In the case of on-road driving, vehicles must stay on the road and as such, the road network provides the constraints dictating the bounds in which a vehicle may travel. A database structure has been developed to capture detailed information about the road network, which includes information about the curvature of lanes, road interconnectivity, signage and traffic control, lane marking, etc. Equations representing the path of the roads can be inferred from the information in the database, and these equations serve as the basis for representing the possible paths the vehicle may take along the road network. Details about the database will be the topic of a future paper.

3.1. Discretizing Actions

The rule-based prediction approach requires that you discretize the possible actions that a moving object may take. In the case of a vehicle driving on-road, we limit the actions of the vehicle to be:

- o Remain at a constant velocity in the current lane
- o Slowly accelerate in the current lane
- o Rapidly accelerate in the current lane
- o Slowly decelerate in the current lane
- o Rapidly decelerate in the current lane
- o Change to a lane on the left
- o Change to a lane on the right
- o Turn to a lane on the left (at an intersection)
- o Turn to a lane on the right (at an intersection)
- o Make a U-Turn (at an intersection)

Figure 2 shows an example of a vehicle on a three-lane, one-way road. Each possible discretized action that the vehicle can take in this scenario is shown, along with the probability that the vehicle will take this action (represented by a value between zero and one in parenthesis). The point on the road that is referenced by each action shows the position the vehicle will be at if that action is performed. So, if we assume that the vehicle is at (0,0) to start and moving along its lane at 3 m/s, then the vehicle will be at (1,0) if a rapid deceleration action is performed, at (2,0) if a slow deceleration action is performed, etc.

Figure 3 shows how we can project these actions into the future to predict the position of the vehicle at longer time horizons. At time = t, the vehicle is at location (0,0). To get to time t+1, the vehicle may perform any of the discretized actions. The results of any of these actions will result in the vehicle occupying a location in the environment at one time step in the future (t+1). This location is shown as (x,y) coordinates next to each possible action. The probability that the vehicle will take any one of these actions over another is determined by the influencing factors described in Section 4.

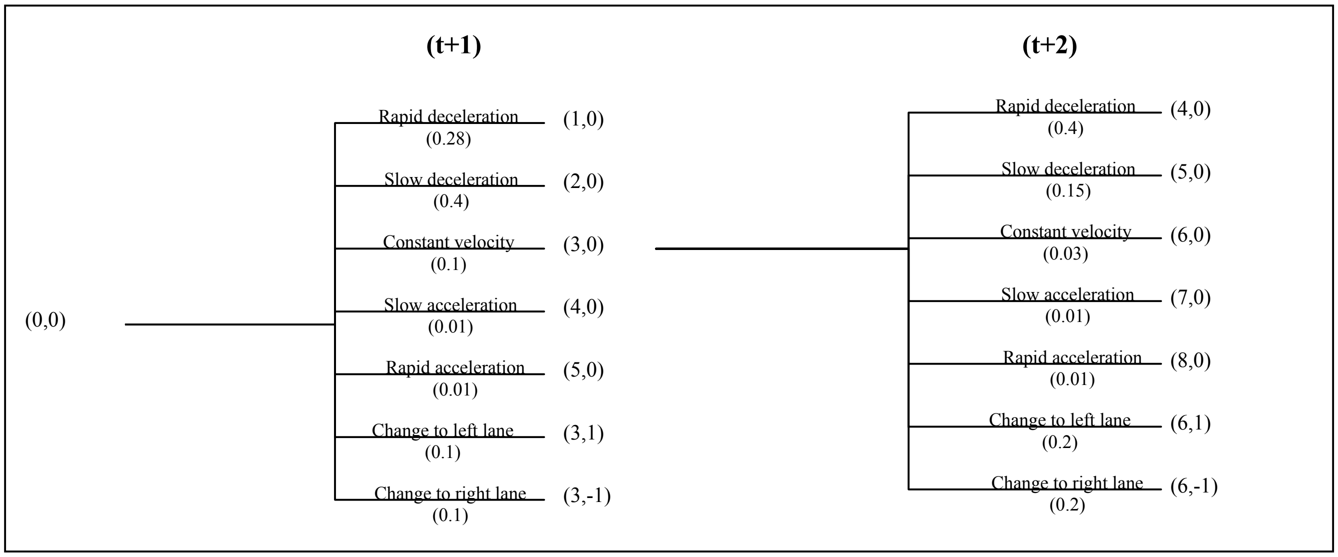

(t+1)
(t+2)
(0,0)
Rapid deceleration (0.28) (1,0)
Slow deceleration (0.4) (2,0)
Constant velocity (0.1) (3,0)
Slow acceleration (0.01) (4,0)
Rapid acceleration (0.01) (5,0)
Change to left lane (0.1) (3,1)
Change to right lane (0.1) (3,-1)
Rapid deceleration (0.4) (4,0)
Slow deceleration (0.15) (5,0)
Constant velocity (0.03) (6,0)
Slow acceleration (0.01) (7,0)
Rapid acceleration (0.01) (8,0)
Change to left lane (0.2) (6,1)
Change to right lane (0.2) (6,-1)

Figure 3: Probabilistic Prediction Over Multiple Time Steps

To get to the next time step (t+2), each of the possible actions that vehicle may take is again determined, and the probabilities are associated based upon the action it took at the previous time step. We continue this into the future as long as our planning horizon requires. Then to determine the probability of the vehicle occupying a given point in space at a given time, we determine if any of the paths result in the vehicle being in that location, and multiply together all of the probabilities along that tree branch to determine the overall probability.

In this case, we are assuming that the vehicle is driving on a straight, horizontal road and as such, the vehicle's location is simply moving in the x-direction. In reality, the location that the vehicle occupies will be derived from the information stored in an *a priori* road network database being developed at NIST.

3.2. Reducing Computation Time

One issue that arises with this approach is the possibility of a large amount of information that needs to be captured. If we have ten possible actions and we are projecting out 20 time steps into the future, we have 10^{20} values that need to be computed. This is an unrealistic expectation for any system that is expected to run in real-limiting the actions a vehicle may take at a given location, and then by limiting the time horizon of the prediction.

First, it is often the case that only a subset of the actions that the vehicle may take is possible at a given location on the road. For example, if the vehicle is not at an intersection, turning right, turning left, and making a U-turn is not possible. If the vehicle is driving in the right lane of a road, it is not possible for it to change to the right lane. If the vehicle is stopped, it may not perform either of the two deceleration actions. These limitations greatly limit the number of actions that need to be represented at any given time. In Figure 2, since the vehicle is not at an intersection, the turn right, turn left, and make a U-turn activities are not listed. It is expected that there will be between 5 and 7 possible actions, on average, at a typical position on the map.

Second, we will initially be implementing these algorithms in the 4D/RCS architecture [1]. 4D/RCS is a hierarchical architecture and limits the planning time horizon at each level of the architecture. Plans at each level typically have 5 to 10 steps between the anticipated starting state and a planned goal state at the planning horizon [1].

Third, we may wish to prune the tree from the onset to eliminate actions that have a very low probability of occurring. For example, if we assume that the probability that a vehicle will rapidly accelerate at time = t+1 is less than a certain percentage (say, 3 percent) then we may decide to ignore that action in the tree. By doing this, we would also ignore all branches of this tree that would follow from this action taking place, thus greatly reducing the size of the tree.

Considering these three factors, the number of values that need to be computed is greatly reduced (from 10^{20} to as little as 5^5) and as such, we believe that this approach should lend itself to real-time environments.

4 Constraints on Motion and Influencing Factors

This section discusses the factors that affect the probabilities associated with the possible actions that a vehicle may take while driving on-road. There are two classes of factors that we must consider. The first are factors that limit the possibilities of where the vehicle is *able* to reach. In other words, by considering these factors, we can eliminate certain portions on the maps that are not reachable by the vehicle. We call these 'constraints on motion'. The second are factors that influence which of the possible actions the vehicle *is likely to perform* out of those that are available to it. We call these influencing factors. These two categories of information are discussed below.

4.1 Constraints on Motion

As mentioned above, the constraints on motion limit the possibilities of the locations that the vehicle is able to reach. Below we discuss two constraints on motion:

- **A Priori Road Network Information:** Assuming that the vehicle is driving on-road and will remain on-road, the road network limits the possible locations that the vehicle can possibly attain.

- **Vehicle's Motion Capabilities:** Motion capabilities of a vehicle limit the possibilities of where it can possibly be in the future. For example, the vehicle's acceleration capabilities restricts the range of locations that are accessible by the vehicle in a given timeframe. Similarly, knowing a vehicle's minimum turning diameter as a function of its current velocity provides a limitation on how quickly it can change lanes and its ability to perform turns at an intersection. One of the ways this information may be used is to limit the possibility of a vehicle turning at an intersection as a function of its velocity approaching the intersection. That is, if a vehicle is approaching an intersection at a relatively high rate of speed, one may eliminate the possibility that the vehicle is turning at the intersection.

4.2 Influencing Factors

Influencing factors affect the probability that a vehicle will perform one action over another. Seven influencing factors are discussed below:

- o **Weather and Environmental Conditions:** Weather and environmental conditions include rain, sleet, snow, fog, darkness, etc. and their effects on visibility and slickness of the road surfaces. As the weather and environmental conditions worsen, the probability often increases that the vehicle's velocity will decrease. Also in these conditions, vehicles often prefer to remain in their lane as opposed to switching lanes or performing passing maneuvers.

- o **Vehicle's Intention and Indicators:** One of the strongest factors that play a role in human's ability to predict the future location of another vehicle is the vehicle's perceived intentions. Intention could be known *a priori*, such as knowing a vehicle is driving to the bank, and this knowledge could be used to determine the most probable path it will take to achieve that goal. More commonly, intentions could be derived from perception, such an indication that a vehicle is making a left turn based upon the vehicle moving into a turn lane or having its blinker on. As more information becomes available from the vehicle, this information can be used to either strengthen or weaken the perceived intentions, which in turn would increase/decrease the probabilities associated with the possible actions the vehicle may take in the future.

- o **Class of the Vehicle:** Object classification provides information about the class of object that is being perceived. If we limit our scope to vehicles on the road, the class of vehicle could indicate the course the vehicle is expected to travel, or how it is expected to behave in certain situations. For example, if the vehicle was identified as being a city bus, we would most likely expect it to stop at a bus stop signs, and traverse primarily in the right most lane. Similarly, if the vehicle was identified as an emergency vehicle, we would expect it to travel at high rates of speed and not to necessarily stop at stop signs and traffic lights. If the vehicle was a motorcycle, we may not eliminate the possibility of it navigating in between vehicles stopped in a traffic jam.

- o **Vehicle's Personality:** When humans drive on-road, they implicitly assign a personality to other vehicles. For example, if one sees another vehicle swerving in and out of traffic, and making unsafe lane maneuvers, one may assign a very aggressive personality to the vehicle. Conversely, if a vehicle is observed driving at or below the speed limit, keeping an extraordinarily far following distance, and rarely changing lanes, a low level of aggressiveness would be assigned. Based on these personality measures, one would expect different actions from that vehicle in the future. For example, an aggressive vehicle would be more prone to make lane changes, and as such, the probability assigned to the action of changing lanes would be greater for that type of vehicle.

- o **Traffic Control Indicators / Rules of the Road:** The 'rules of the road' play a large role in predicting how a vehicle is expected to behave under certain situations. For example, if a vehicle is approaching a stop sign, one would expect that the vehicle would gradually decrease its speed until it reaches the stop sign, comes to a complete stop, and then proceeds when the intersection is safe to traverse. However, based on the perceived personality of the vehicle, we may expect that the vehicle only slows down but does not come to a complete stop, or traverses the intersection before most would consider it safe.

  Efforts at NIST have focused on encoding the rules of the road using finite state machines leveraging a driver's manual published by the Department of Transportation [7]. The document contains a comprehensive inventory of the behaviors involved in operating an automobile, along with the rated criticalities of these behaviors. The task descriptions are organized in terms of the situations giving rise to the behaviors; behaviors involved in controlling movement of the car without regard to specific situations; behaviors that must be performed continually or periodically while driving, rather then in response to a specific situation; and off-road behaviors that are performed before driving, to maintain the car in sound operating condition, and in compliance with legal regulations. The document organizes the task descriptions into the following categories:

  - o basic control (situation-independent driving behaviors to control the movement of the vehicle),
  - o general driving (continuously-performed driving behaviors in response to any specific situation),
  - o situational behaviors (behaviors that are required in response to specific situations),

- pre-driving behaviors (behaviors taken prior to driving to assure safe and efficient operation),
- maintenance (behaviors directed toward the vehicle to assure safe and efficient operation), and
- legal responsibilities (legally imposed behaviors required to assure that drivers are responsible for the consequences of their actions).

- **Other Traffic:** In the same way that our vehicle is predicting the future locations of other vehicles in its vicinity, other vehicles are doing the same with vehicles in their vicinity. Hence, our vehicle needs to not only be cognizant of vehicles that run a risk of interfering with our path, but also of vehicles that could interfere with those vehicles' paths. This is analogous to a driver looking two cars ahead to try to predict what the car in front is going to do.

  Each vehicle on the road has a range of influence associated with it. The size of this range is a function of the vehicle's velocity and the presence of intersections, among other factors. Any vehicle within a defined range could be impacted by actions in which that vehicle performs. This, in turn, could cause a ripple effect. As shown in figure 4, Vehicle B is in Vehicle A's range of influence (denoted by the right-most oval). Similarly, Vehicle C is in Vehicle B's range of influence (denoted by the left-most oval). Even though Vehicle C is not in Vehicle A's range, Vehicle C still needs to be aware of Vehicle A's motions since these motions will affect vehicle B which in turn will affect Vehicle C.

  Information about the position and motion of other traffic will affect the probability of other vehicles taking certain actions. Constraints, such as maintaining safe following distance, play a strong factor in how a vehicle reacts to certain situations.

- **Formations:** Formations aren't as important for on-road driving as they are for off-road driving, but they are still worth mentioning here. If it can be determined that a vehicle is driving as a part of a larger formation, the rules that govern the formation play a large role in dictating where that vehicle will be in the future. In the case of a battlefield environment, the military has devised a number of formations that vehicles in a group follow (e.g., bounding overwatch, V-formation, etc.). Similarly, in Robocup competition, teams often implement different strategies that rely on different formations. Knowing the other team's strategy can help to predict the players' moves. Even in on-road environments, vehicles sometimes move in formations, such as funeral processions. Identifying a presidential procession can provide additional information about the future moves of the vehicles in the procession. For example, the vehicles in the

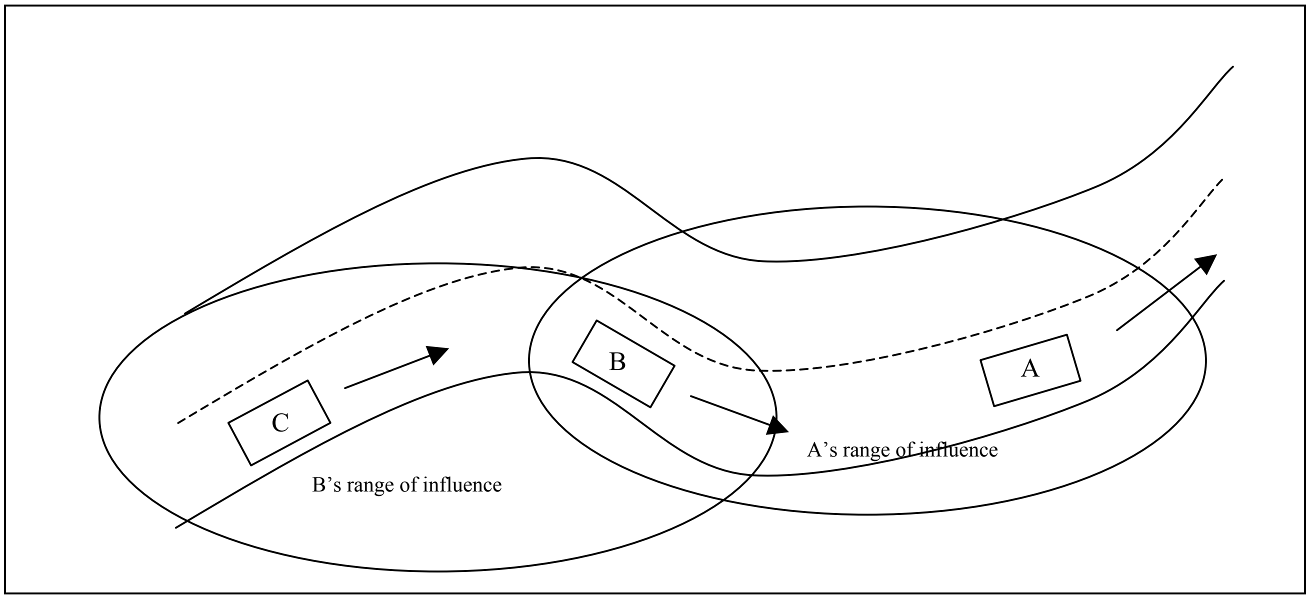


Figure 4: Vehicle's Range of Influence

procession will most likely change lanes when the vehicle in front of them changes lanes. Also, the vehicles will likely run red lights to keep up with the vehicle in front of them.

The factors mentioned above provide much of the input necessary to determine and refine the probabilities that predict the future location of moving objects in the environment. This information is then fed to the planners in the form of space/time probability distribution (in the planner's formalism of choice) to develop appropriate plans in the presence of moving objects.

5 Planning in a Dynamic Environment

As described in [2], the NIST planner utilizes incrementally created planning graphs to formulate potential vehicle trajectories. As part of the graph expansion/evaluation phase, a cost/benefit number must be assigned to each potential path segment. The dynamic obstacle layer of the planner's world model system determines a portion of this cost/benefit number.

If the trajectory of the moving object is known explicitly, the moving object prediction subsystem would produce a curve through space and time that represents the path of the moving object. The dynamic obstacle layer would then match this curve against the plan segment being evaluated to determine if an intersection exists. This collision information is passed onto a value judgment module that examines the predicted nature of the obstacle (e.g. is it a soda can or a tank) and the intent of the commander (e.g. allowed to run over soda cans, but not tanks) in order to formulate the dynamic obstacle portion of the overall cost/benefit number for the plan segment.

In the real world, moving obstacles seldom broadcast their exact trajectory ahead of time and predictive algorithms are necessary to compute a potential trajectory. This potential trajectory is made available to the dynamic obstacle layer in the form of equations that represent a volume in space/time for the expected location of the object. The volume is often a very tight circle at the current time (where the location of the vehicle is known with small uncertainty), and the size of the volume per unit time will gradually increase as one moves forward in time. This increase in volume represents the uncertainty in the location prediction. For a ground-based object, this bounding area may be viewed as a three-dimensional volume with axes of northing, easting, and time. The dynamic obstacle layer must now examine if a potential path segment lies inside the volume that represents any value over a pre-defined probability threshold. This information is sent to the value judgment module for use in the formulation of the final cost/benefit number. Through the use of this system, minimal collision or collision free paths may be planned.

6 Conclusion

In this paper, we have presented an overview of a framework for representing and planning the future location of moving objects. In our research, we quickly found that there was a clear void in the literature in areas focusing on long-range prediction of moving objects in a constrained environment. As such, we have developed an approach to predict the future location of objects in a constrained environment.

The approach explores applying probabilistic, logic-based algorithms to predict the future location of vehicles in an on-road environment. To apply this approach, the possible motions of the object are discretized and each action is assigned a probability based upon a series of 'constraints on motion' and influencing factors that are described at a high-level in this paper. Although these factors may not be exhaustive, we believe that they provide a good representative sample of the types of factors that would need to be applied.

The concepts in this approach are very new and a number of issues still need to be explored. We need to work out the details on how the influencing factors will contribute to the probabilities associated with the discretized actions. This paper discussed the contributions at a coarse level.

We also need to ensure that this approach can be performed in a real-time environment. Although we have proposed pruning mechanisms to curtail the unbounded growth of the probability trees, we still need to ensure that the pruned tree can still be developed and processed in the time constraints imposed on the planners.

Before we use this approach, we need to decide which moving objects to apply it to. In the case of on-road driving, there are many moving objects in the environment, but not all of them need to have a detailed level of prediction associated with them. Only the vehicles that have the highest probability to affecting our path would be of concern. For other vehicles, we may employ a less accurate and a less computationally expensive approach.

Although this approach was only applied to on-road driving in this paper, it would be equally applicable to any other type of moving object provided that the actions of the object can be discretized. Future work will apply this technique to pedestrians and military vehicles.

References

1. Albus, J. and et.al., "4D/RCS Version 2.0: A Reference Model Architecture for Unmanned Vehicle Systems," NISTIR 6910, National Institute of Standards and Technology, Gaithersburg, MD, 2002.

2. Balakirsky, S. and Herzog, O., "Planning with Incrementally Created Graphs," NIST, 6895, Gaithersburg, MD, 2002.

3. Bar-Shalom, Y. and Fortmann, T. E., *Tracking and Data Association*, Academic Press 1988.

4. Firby, J., "Architecture, Representation, and Integration: An Example from Robot Navigation," *Proceedings of the 1994 AAAI Fall Symposium Series Workshop on the Control of the Physical World by Intelligent Agents*, New Orleans, LA, 1994.

5. Gueting, R. H., "A Foundation for Representing and Querying Moving Objects," *ACM Transactions on Database Systems (TODS)*, Vol. 25, No. 1, 2000, pp. 1-42.

6. Madhavan, R. and Schlenoff, C., "Moving Object Prediction and Tracking for Off-road Autonomous Navigation," *Proceedings of the SPIE Aerosense 2003 Conference*, Orlando, FL, 2003.

7. McKnight, J. and Adams, B., *Driver Education Task Analysis. Volume 1. Task Descriptions*, Human Resource Research Organization, Department of Transportation, National Highway Safety Bureau 1970.

8. Schlenoff, C., "Linking Sensed Images to an Ontology of Obstacles to Aid in Autonomous Driving," *Proceedings of the 18th National Conference on Artificial Intelligence: Workshop on Ontologies for the Semantic Web*, 2002.

9. Schlenoff, C., Madhavan, R., and Balakirsky, S., "Representing Dynamic Environments for Autonomouos Ground Vehicle Navigation," *Submitted to the IEEE/RSJ IROS 2003 Conference*, Las Vegas, NV, 2003.

10. Singhal, A., *Issues in Autonomous Mobile Robot Navigation*, Computer Science Dept, U. of Rochester 1997.

11. Stone, L., Barlow, C. A., and Corwin, T. L., *Bayesian Multiple Target Tracking*, Artech House 1999.

On Planning for Multi-Agent Opportunistic Execution

Carmel Domshlak
Dept. of Computer Science
Cornell University
Ithaca, NY 14850, USA
dcarmel@cs.cornell.edu

James H. Lawton
Information Directorate
US Air Force Research Laboratory
Rome, NY 13441, USA
lawton@ai.rl.af.mil

Abstract

We examine multi-agent systems of planning agents, where the resource consumption of the agents' actions is uncertain. For such systems we introduce a model of planning and execution that treats differently the qualitative (in our case, certain), and quantitative (partly uncertain) effects of the agent actions. The planning stage mostly addresses the qualitative part of the problem, while the execution takes a form of an approximate decision-theoretic approach. During the execution, our model allows the agent to respond opportunistically to the changes in its environment, even if no re-planning is possible. In particular, our model provides a flexible platform for multi-agent opportunistic assistance, such that the latter can be even more efficient for the agent than adjusting its own behavior.

1 Introduction

During the last decade, research in classical AI planning has mostly concentrated on problems involving cascading levels of action selection with complicated logical interactions between actions, while making strong assumptions about time, resources, and the objective guiding the planning [Smith *et al.*, 2000]. While this effort has lead to enormous success [Geffner, 2002], the increasing emphasis on real-world applications has led AI planning researchers to develop algorithms and systems that more closely match realistic environments. In such real-world environments, the planning activity is often distributed and continual (i.e. planning and execution are interleaved) [desJardins *et al.*, 1999], problem specification often involves uncertainty about action duration and resource consumption [Bresina *et al.*, 2002], and the objective is specified using a non-trivial preference model.

In such complex environments, the planning and execution context might change in ways that suggest a change in plans. Unexpected changes in the world might provide *opportunities* to either accomplish goals more effectively or to reconsider the choice of goals that the agent is attempting to achieve. Alternatively, the agent's goals and/or abilities might change, so that although the current plan could still be carried out to some degree, the motivation for doing so may have decreased, while some secondary courses of action may have become more attractive than they were before. Finally, in a multi-agent environment, each agent should adapt itself to the activities of other agents. Thus, when various aspects of the world can evolve continuously throughout a pre-planned episode of execution, an agent should continuously evaluate and revise its plans.

One of the main computational concerns with such systems is the complexity of planning. Planning is known to be intractable even for models with extremely severe formalism limitations [Bylander, 1994], and extending the formalism to capture more realistic scenarios makes it even harder. In addition, in certain planning domains, such as groups of autonomous Mars rovers [Washington *et al.*, 1999] or controllers for complicated hardware systems [Williams and Nayak, 1997], many actions are potentially risky and require pre-approval by mission operations personnel. In such domains, online re-planning by an agent as a way to revise its current plan to adapt to opportunities and/or failures in its multi-agent group can be too costly, too difficult, and too risky. Following this observation, the first question that we attempt to address in this paper is: *To what extent can planning agents opportunistically improve their own behavior, and assist other agents, in systems where little or no re-planning is possible?* In other words, how far can we take the planning stage of the system such that plan execution will be opportunistic at both single- and multi-agent levels without the need for re-planning?

An additional aspect of the problem that we address in this paper is the practical complexity of extending single-agent opportunism to multi-agent opportunism. While *single-agent opportunism* refers to the ability of an agent to alter a pre-planned course of action to pursue a different set of goals, based upon a change in the environment or in the agent's internal state (an opportunity) [Hammond, 1993; Lawton, 1999], *multi-agent opportunism* is the ability of agents in a multi-agent system to assist one another by recognizing and responding to potential opportunities for each other's goals. Our intention is to provide a system of multiple planning agents with a multi-agent opportunistic behavior via the *same mechanisms* used to provide it with a single-agent opportunistic behavior. In this paper we show that such mechanisms can be constructed even for systems where re-planning is undesirable, and provide an example of such a mechanism for a concrete planning and execution model. In addition, and somewhat surprisingly, we show that *assistance to another agent can be even more efficient than reconsidering one's own course of action*. Therefore, in some cases, supporting multi-agent opportunism can be even more cost-effective than supporting single-agent opportunism.

2 Abstract Model of Multi-Agent Planning and Execution

The abstract model we are using for representing a multi-agent system is very similar to models used in [Shehory and Kraus, 1996] and [Ogston and Vassiliadis, 2001], but is extended to express opportunistic behavior. We model a multi-agent system as a collection of benevolent agents $\{\mathbb{A}_1, \cdots, \mathbb{A}_n\}$, where each agent $\mathbb{A}_i$ is associated with a set of capabilities $C_i = \{c_{i_1}, \cdots, c_{i_l}\}$, and a set of resources $R_i = \{r_{1_i}, \cdots, r_{m_i}\}$. For $1 \leq j \leq m$, we have $r_{j_i} \in \mathit{Dom}(\mathbf{r}_j)$, where $\mathbf{r}_j$ is a certain type of resource (e.g., time, energy, etc.), and $\mathit{Dom}(\mathbf{r}_j)$ is the corresponding, possibly continuous domain of the resource type. Note that in the related models mentioned above, the difference between the capabilities and the resources is not very clear. To clarify this point, in our model *the capabilities of an agent $\mathbb{A}_i$ correspond to the goals that in general can be assigned to $\mathbb{A}_i$*. For example, consider a team of three planetary rovers $\mathbb{A}_1$, $\mathbb{A}_2$ and $\mathbb{A}_3$, where both $\mathbb{A}_1$ and $\mathbb{A}_2$ are equipped with cameras, while $\mathbb{A}_3$ is not. In this group, the goal "have picture of location `L1`" is in both capabilities sets C_1 and C_2, but not in C_3.

In addition to the acting agents $\{\mathbb{A}_1, \cdots, \mathbb{A}_n\}$, we assume there is an abstract agent $\mathbb{B}$ acting as a *task broker* [Klusch and Sycara, 2001]. We use $\mathbb{B}$ to simplify the description of the information flow in the system: The primary job of $\mathbb{B}$ is simply to dispatch the goals of the system to the various agents. The decision process behind $\mathbb{B}$, as well as its actual implementation, are not within the scope of this work. The only thing we assume about $\mathbb{B}$ is that if $\mathbb{B}$ assigns goal g to agent $\mathbb{A}_i$, then we have $g \in C_i$.

Traditionally, given a set of goals $G_i = \{g_{i_1}, \cdots, g_{i_k}\} \subseteq C_i$, agent $\mathbb{A}_i$ plans for this set of goals, and begins the execution of the generated plan $\mathcal{P}$. To support better realistic domains, we assume that each goal g is annotated with its value V_g (which may be parameterized by various parameters such as deadlines, energy consumed, etc.), and that the planning process takes these value functions into account. Likewise, each action is associated with its resource consumption, which in most practical domains will not be certain, and thus will be represented using a resource consumption distribution.

During the execution of a plan $\mathcal{P}$ by some agent $\mathbb{A}_i$, several aspects of the world could change, impacting the relative attractiveness of $\mathcal{P}$. For instance, any of the following may occur:

(a) $\mathbb{A}_i$ is assigned an additional goal $g_{i_{k+1}}$ by $\mathbb{B}$.

(b) Some other agent $\mathbb{A}_j$ in the group fails to accomplish one of its assigned goals $g \in G_j$.

(c) The value V_g for some $g \in G_i$ has been changed (positively or negatively).

(d) Some of the goals in G_i becomes unreachable with respect to $\mathcal{P}$.

(e) Resource consumption by the part of $\mathcal{P}$ executed so far has been significantly different (positively or negatively) from what it was expected during planning.

In such cases, $\mathbb{A}_i$ should revisit its current course of action, possibly updating its set of active goals, and *suspending* goals it determines are no longer feasible. Normally, these suspended goals are returned to the broker for redistribution to other agents in the multi-agent system. In our model, though, $\mathbb{A}_i$ may attempt to satisfy these goals opportunistically by fitting them into its current plan, or into the current plan of another agent in the multi-agent system, without re-planning.

Our aim has been to come up with a concrete model of plan execution that will support flexible, opportunistic behavior at both the single- and multi-agent levels, while requiring no re-planning of the qualitative part of the plan, i.e. the part of the plan that encodes the interactions between various actions of the plan. In some sense, we would like our re-planning to take a form of *reasoning about the plan*, rather than actual re-planning. To what degree and how this mission can be accomplished is discussed in the rest of the paper.

3 Basic Model for Planning and Execution

In the area of AI planning, problems containing actions with uncertain effects have mostly been modeled using the standard decision-theoretic tools, such as Markov decision processes [Blythe, 1998; Boutilier *et al.*, 1999]. However, in cases where there is uncertainty about quantitative consequences of the actions (such as consumption of various resources), these tools require covering the space of the availability of all possible resources. Thus, it is very unlikely that a standard decision-theoretic planning approach that represents and reasons about all possible decision points would be practical.

Therefore, while modeling a concrete scheme for planning and multi-agent execution, our intention has been to trade optimality at the planning stage for efficiency and a significant degree of flexibility during the actual execution. The basic formalism for specifying the planning problems for each agent has been significantly inspired by the work on contingency planning for planetary rovers [Dearden *et al.*, 2002]. Our main motivation for adopting this formalism for problem specification was to stay as close to real-world domains as possible.

3.1 Planning

Following [Dearden *et al.*, 2002], we assume that the qualitative part of the problem is described using the propositional STRIPS formalism in which both positive and negative preconditions are allowed[1] [Bylander, 1994]. Each agent is associated with a description of its initial state (represented as a conjunct of valid propositions), a set of goal propositions

[1]This is exactly the formalism used for the first level of planning competition [Fox and Long, 2002b].

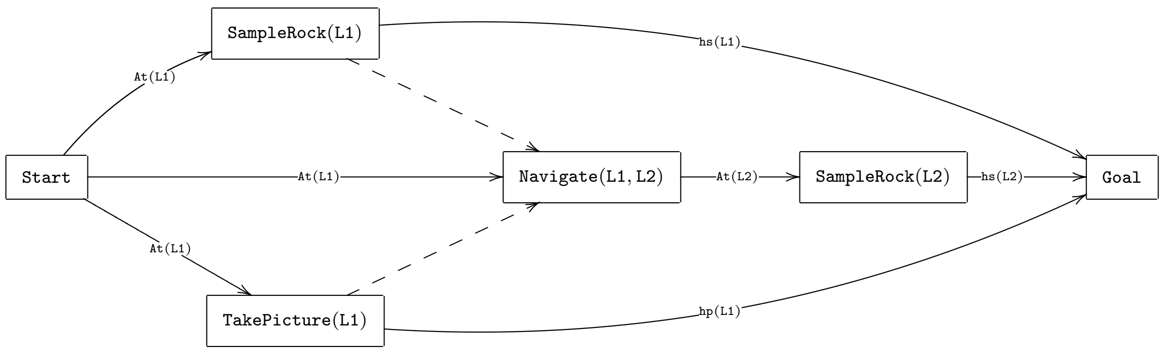


Figure 1: Partial order plan for the running example.

to be achieved, and set of possible actions, each of which is characterized by its preconditions and effects. In what follows, we denote the preconditions and effects of action A by $\mathsf{prec}(A)$ and $\mathsf{effects}(A)$, respectively.

The quantitative part of the problem is described by the resource consumptions of the actions and the values associated with each goal. Resource consumption is modeled using consumption distributions associated with each action. Goal values are modeled as functions of the resources available after achieving these goals. For example, if $\mathbf{r}$ is the only resource used by the agent, and the value function $V_g(\mathbf{r})$ of goal g is:

$$V_g(\mathbf{r}) = \begin{cases} 0, & \mathbf{r} \leq 0 \\ 10, & \mathbf{r} > 0 \end{cases}$$

then the value of g is 10 if we can achieve g with some resource remaining, and 0, otherwise.

As with [Dearden *et al.*, 2002], in the planning stage we ignore the quantitative part of the problem, solving the STRIPS-based problem as if there is no resource consumption or difference in the importance of the goals whatsoever. However, the difference between our model of planning and that in [Dearden *et al.*, 2002] is that the latter corresponds to the first stage of the Graphplan algorithm [Blum and Furst, 1997], resulting in a *plan graph*, while we are interested in a structure that has properties of both a plan graph and a *partial order plan* [McAllester and Rosenblitt, 1991]. Later we justify why plan graphs alone will not serve us properly.

First, we specify the notions of partial order plans and its slight extension we are using in our work. A partial order plan is a tuple $\langle \mathcal{A}, \mathcal{O}, \mathcal{L} \rangle$, where $\mathcal{A}$ is a set of actions, $\mathcal{O}$ is a set of *ordering constraints* over $\mathcal{A}$, and $\mathcal{L}$ is a set of *causal links*. For example, if $\mathcal{A} = \{A_1, A_2, A_3\}$ then $\mathcal{O}$ might be the set $\{A_1 < A_3, A_2 < A_3\}$. These constraints specify a plan in which A_3 is necessarily the last operator, but do not commit to a particular order on A_1 and A_2. Naturally, the set of ordering constraints must be consistent, i.e. there must exist some total order satisfying them. A causal link has the form $A_i \xrightarrow{q} A_j$, where A_i and A_j are actions and q is a proposition. Such a causal link denotes the fact that A_i produces (i.e. has the effect) q which is consumed by A_j (i.e. used to satisfy a precondition of A_j). Ordering constraints are imposed among the actions to ensure that other actions do not threaten the causal links.

For example, consider a simple problem that is based on the `Rovers` domain used in the recent planning competition [Fox and Long, 2002b]. Three operators available to the agent are:

$\mathtt{SampleRock(p)}$
  PRECONDITIONS:
    $\mathtt{location(p)} \wedge \mathtt{At(p)}$
  EFFECTS:
    $\mathtt{hs(p)}$

$\mathtt{TakePicture(p)}$
  PRECONDITIONS:
    $\mathtt{location(p)} \wedge \mathtt{At(p)}$
  EFFECTS:
    $\mathtt{hp(p)}$

$\mathtt{Navigate(p, q)}$
  PRECONDITIONS:
    $\mathtt{location(p)} \wedge \mathtt{location(q)} \wedge \mathtt{At(p)}$
  EFFECTS:
    $\neg\mathtt{At(p)} \wedge \mathtt{At(q)}$

where the propositions $\mathtt{hs(p)}$ and $\mathtt{hp(p)}$ stand for "have rock sample" and "have picture" from location $\mathtt{p}$, respectively. The (relevant part of the) initial state of the agent is $\mathtt{At(L1)} \wedge \neg\mathtt{hs(L1)} \wedge \neg\mathtt{hp(L1)} \wedge \neg\mathtt{hs(L2)}$, while the goals are $\mathtt{hs(L1)}$, $\mathtt{hp(L1)}$, and $\mathtt{hs(L2)}$. Figure 1 presents (the relevant part of) a possible partial order plan for this problem, where the solid edges represent the causal links (labeled with the corresponding propositions), and the dashed edges represent the ordering constraints that are not trivially entailed by the causal links[2]. It is easy to see that there are two totally ordered plans consistent with this partial order plan, and the only difference between them is the relative positions of $\mathtt{SampleRock(L1)}$ and $\mathtt{TakePicture(L1)}$.

After constructing a partial order plan for the problem, we slightly change its structure, making it reminiscent of the plan graphs generated by the Graphplan-based algorithms. We refer to this structure as a *partial order plan graph* (POPG, for short). The POPG for the running example is depicted in Fig-

[2]`Start` and `Goals` nodes are dummy actions acting as a producer of the initially valid propositions and the consumer of the goal propositions, respectively.

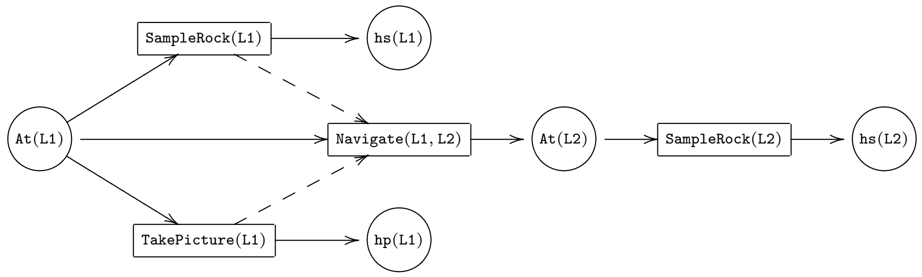


Figure 2: Partial order plan graph (POPG) for the running example.

ure 2. As with a plan graph, a POPG contains two types of nodes: proposition nodes and action nodes. The action nodes are exactly the nodes of the original partial order plan (excluding the dummy start and goal nodes), while each proposition node is created by contracting the causal links corresponding to effectively the same proposition. However, in contrast to plan graphs, a POPG is not a leveled graph, and the alternative schedules of the plan are captured by the ordering constraints of the original partial order plan.

After constructing the "skeleton" plan for the qualitative part of the problem in the form of a POPG, we add the resource consumption distributions and the value functions into this structure: The actions are annotated with their resource consumption distributions and the goal nodes are annotated with their value functions. The resulting structure is ready to be used in the execution stage.

3.2 Execution

Given an initial state, a set of resources, and a POPG structure of the plan enriched by the quantitative information about resource consumption and values of the different goals, the agent starts to execute its plan. At each intermediate state s during the execution, the agent should make a decision about the next action to perform. As the agent is provided with a partial order plan, there may be more than one action applicable in the state s, given the current set of resources. For instance, in the initial state of the running example, both actions $\mathtt{SampleRock}(\mathtt{L1})$ and $\mathtt{TakePicture}(\mathtt{L2})$ are consistent with the partial order plan represented by the POPG in Figure 2. In addition, however, observe that the action $\mathtt{Navigate}(\mathtt{L1},\mathtt{L2})$ can be performed in the initial state as well. Clearly, if resources are not an effective limitation, performing this action will be irrational, as the agent will loose its ability to achieve the goals $\mathtt{hs}(\mathtt{L1})$ and $\mathtt{hp}(\mathtt{L1})$. On the other hand, if the resources are limited and the goal $\mathtt{hs}(\mathtt{L2})$ is very important, it might be the case that the right thing to do is to forget about $\mathtt{hs}(\mathtt{L1})$ and $\mathtt{hp}(\mathtt{L1})$, and to perform $\mathtt{Navigate}(\mathtt{L1},\mathtt{L2})$, trying to achieve $\mathtt{hs}(\mathtt{L2})$ with as little risk as possible. Having an ability to easily access and reason about *all* the applicable actions available in the plan has motivated us to adopt partial order plans, and not, for instance, Graphplan-based plan graphs in which for the same purpose we will have to perform a less straightforward analysis of the graph's levels.

In general, for an agent to decide which action among a set of alternative actions should be performed, requires an estimate of how much value could be gained by performing each of these action. Computing these values exactly is intractable, as it requires taking into account not only the probability of certain resource consumption by each action to be executed in the future, but also capturing in the model all possible results of potential future failures. However, adopting the way the resource consumption distributions are abstracted in [Dearden *et al.*, 2002], we outline an approximation method for such a value estimation.

For ease of presentation, in what follows we assume, without loss of generality, that there is only one resource, and ρ stands for the amount of this resource available at the decision point. When estimating the value of performing a given action, instead of taking into account the precise resource consumption distributions, each action A is annotated with (i) its expected consumption $\mu(A)$ and (ii) the minimal resource level $\min(A)$ required to allow A to be executed. This way, if $\rho \geq \min(A)$, then A can be executed, its execution is assumed to be successful, and its resource consumption is expected to be $\mu(A)$. Otherwise, if $\rho < \min(A)$, then the action is not executable, due to the risk associated with its failure. Note that, while $\mu(A)$ is defined purely by the resource consumption distribution of A, $\min(A)$ is specified explicitly and is part of problem modeling.

Let $\mathsf{actions}(\mathcal{P}, s, \rho)$ (specified by Eq. 1) be the set of actions in $\mathcal{P}$ that are executable in state s with ρ amount of resource available.

$$\mathsf{actions}(\mathcal{P}, s, \rho) = \{A \in P \mid \mathsf{prec}(A) \in s \ \wedge \ \rho \geq \min(A)\} \quad (1)$$

The value $U(\mathcal{P}, s, \rho)$ represents our estimate of how much value could be gained by executing plan $\mathcal{P}$ with ρ amount of resource, starting at the state s. This value is specified by Eq. 3 via (i) the value of the plans that the agent will have after performing one of the actions $A \in \mathsf{actions}(\mathcal{P}, s, \rho)$, and (ii) the value of the goals achieved directly by the action A; these quantities are specified in Eq. 2 by $\alpha(\mathcal{P}, A, s, \rho)$ and $\beta(A, \rho)$, respectively. The part of the plan $\mathcal{P}$ (= subgraph of POPG $\mathcal{P}$) remaining after performing action A in state s is constructed by the procedure $\mathsf{Refine}(\mathcal{P}, A, s)$, which appears in Figure 3. The value of each such "sub-plan" generated by the Refine procedure is evaluated with the initial state $\sigma(s, A)$, which results from executing action A in state s, and with the amount of resource that is expected to remain after executing A.

$$\begin{aligned}\alpha(\mathcal{P}, A, s, \rho) &= U\left(\mathsf{Refine}(\mathcal{P}, A, s), \sigma(s, A), \rho - \mu(A)\right)\\ \beta(A, \rho) &= \sum_{g \in \mathsf{effects}(A)} V_g(\rho - \mu(A))\end{aligned} \quad (2)$$

```
Refine($\mathcal{P}, A, s$)
  1. Remove $A$ from $P$, together with all its outgoing edges.
  2. Iteratively remove:
     • All the proposition nodes $p$ (together with their outgoing edges), such that $p \notin \sigma(s, A)$, and the node $p$ has no incoming edges, and
     • All the action nodes $A'$, such that for at least one of the preconditions $q \in \mathsf{prec}(A')$ there is no proposition node associated with $q$ and having an outgoing edge to $A'$.
```

Figure 3: Procedure for updating POPG $\mathcal{P}$ after performing action A.

$$\begin{aligned} U(\emptyset, s, \rho) &= 0 \\ U(\mathcal{P}, s, \rho) &= \max_{A \in \mathsf{actions}(\mathcal{P}, s, \rho)} [\alpha(\mathcal{P}, A, s, \rho) + \beta(A, \rho)] \end{aligned} \quad (3)$$

Returning to the execution, at every decision point s the agent:

1. Eliminates from its current plan $\mathcal{P}$ all the actions A' that are not executable with the current amount of resource ρ, i.e. $\min(A) > \rho$. For each such action, the agent iteratively refines $\mathcal{P}$ using $\mathsf{Refine}(\mathcal{P}, A', s)$.
2. Estimates the value of $\mathcal{P}$ using the value iteration process described by Eq. 3.
3. Chooses one of the actions $A \in \mathsf{actions}(\mathcal{P}, s, \rho)$ that actually provides $U(\mathcal{P}, s, \rho)$.
4. Performs A (resulting in the state $\sigma(s, A)$, and some remaining amount of resource $\rho' \leq \rho$).
5. Updates its plan $\mathcal{P}$ to the result of $\mathsf{Refine}(\mathcal{P}, A, s)$.

To illustrate the process, consider the POPG of the running example (Figure 2), and suppose that the resource consumptions of the actions as abstracted as follows:

A	$\mu(A)$	$\min(A)$
SampleRock(L1)	3	3
TakePicture(L1)	2	2
Navigate(L1, L2)	10	15
SampleRock(L2)	5	7

Likewise, let the value functions of the goals to be constant, but different: $V_{\mathtt{hs(L1)}} = 2$, $V_{\mathtt{hp(L1)}} = 2$, and $V_{\mathtt{hs(L2)}} = 10$. If we have $\rho = 19$, then $U(\mathcal{P}, s, \rho) = 12$ and the action to be executed is TakePicture(L1), as the estimated best course of action is to perform first TakePicture(L1), then Navigate(L1, L2), and finally SampleRock(L2). However, if $\rho = 18$, then $U(\mathcal{P}, s, \rho) = 10$ and the action to be executed is Navigate(L1, L2), as we estimate that performing any other action will prevent us from achieving hs(L2).

Now, let us examine the flexibility of our planning/execution model with respect to the various possible changes in the environment that we listed in Section 2. First, sudden unreachability of goals, as well as uncertainty in resource consumption by the agent's actions, is captured by the model implicitly. Second, if the value of some of the (still reachable) goals that the agent had planned to achieve have changed, the only thing that the agent should do is to update the value functions associated in its plan with the corresponding goals. All the subsequent decisions will implicitly take into account this change in the agent's objectives. In particular, if one of the agent's goals, g, becomes completely irrelevant (i.e. $V_g = 0$), the agent could easily update its POPG by removing the node g, along with exactly those action nodes that are used to "produce" this node g and are not used to produce any other goal. However, in the next section we show that the latter is not necessarily the best way to handle such situations.

The only part of dynamics that still seems to be problematic is assigning a new goal to an agent (i.e. a goal that is not captured by the current plan $\mathcal{P}$). Such a goal can be either completely new to the multi-agent group, or one of the goals that has been suspended by some other agent in the group. Clearly, a complete re-planning for the extended set of goals will solve the problem, and in many domains such a painful solution might be unavoidable. However, in the next section we argue that, at least for some practical domains, we can slightly extend the above model of planning and execution in a way that re-planning can often be avoided.

4 Planning for Capabilities

Considering the above scheme for planning and execution, our first observation is that nothing prevents us from planning for goals that have no value assigned to them. Similarly, if one of the goals that the agent has planned for becomes irrelevant, instead of removing this goal from the plan, we could simply zero its value function. Since the decision mechanism behind the execution takes into account not only the value of the goals to be achieved, but also the risk behind this or another course of action (expressed via cumulative resource consumption), achieving a goal with a zero value will automatically be postponed.

Recall that in our abstract model of multi-agent systems, each agent is characterized by a set of capabilities that represent the goals that the agent can possibly be assigned. Therefore, instead of planning for the set of goals that the agent has been actually assigned to, one can consider *to plan for the whole set of capabilities* and to reason about the best course of action during the execution, when the value of different capabilities is known better than during the off-line planning. Clearly, the reader may rightfully say that the whole set of capabilities may be huge, and even its explicit description may be intractable. Although we agree that in general nothing prevents the set of capabilities from being orders of magnitude larger than an average set of goals the agent is actually assigned, at least in some domains this does not seem to be the case. For instance, consider a group of planetary rovers that are constructed for fulfill some tasks on Mars [Multi-Rovers, 2002]. At least at this stage of planetary rovers development, the superset of goals that each rover can be assigned does not seem to be too large, yet this domain poses a lot of challenging research and development issues.

Observe that, if planning for capabilities is considered to be

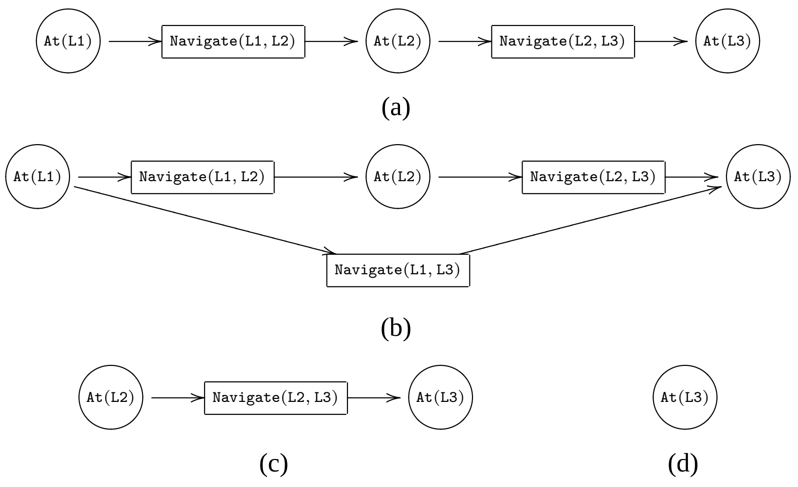


Figure 4: Extending the "seed" POPG: (a) The initially constructed POPG $\mathcal{P}$; (b) EPOPG $\mathcal{P}_e$, resulting from extending $\mathcal{P}$ by an alternative course of action $\mathtt{Navigate}(\mathtt{L1},\mathtt{L3})$; (c-d) The result of Refine on $\mathcal{P}_e$ and the actions $\mathtt{Navigate}(\mathtt{L1},\mathtt{L2})$ and $\mathtt{Navigate}(\mathtt{L2},\mathtt{L3})$, respectively.

as feasible as planning for the actual goals, technically nothing should be changed in the above scheme of planning and execution. However, the plan generated for the whole set of capabilities can be far from efficient with respect to only the actual goals that the agent is assigned to. For instance, suppose that a rover located at location $\mathtt{L1}$ has been assigned to a single goal of sampling a rock in location $\mathtt{L1000}$. If this particular rover is capable of sampling rocks at any of the thousand locations $\mathtt{L1}, \mathtt{L2}, \ldots, \mathtt{L1000}$, the constructed plan may take the rover from $\mathtt{L1}$ to $\mathtt{L1000}$ through all the locations in between, as if preparing this rover to perform the other rock samples as well.

At first view, the above observation seems to point to a serious drawback of planning for capabilities instead of for goals. However, recall that in our scheme the execution is not a blind process: the agent already deliberates between different courses of actions, possibly suspending these or other goals. Therefore, by putting more effort in the planning phase this drawback can be overcome.

4.1 Extended Partial Order Plan Graphs

First, observe that nothing prevents us from enriching the constructed POPG $\mathcal{P}$ of an agent with some actions that might be *inconsistent (cannot be merged together) with each and all possible complete execution of* $\mathcal{P}$. For example, consider a (partial) POPG $\mathcal{P}$ depicted in Figure 4(a). In Figure 4(b) this valid "seed" plan $\mathcal{P}$ is extended to a new structure $\mathcal{P}_e \supset \mathcal{P}$ by adding an action $\mathtt{Navigate}(\mathtt{L1},\mathtt{L3})$. In general, such an extension Υ can be any valid POPG such that the root nodes (i.e. nodes with no incoming edges) of Υ are a subset of the proposition nodes of $\mathcal{P}$ (i.e. Υ is grounded in $\mathcal{P}$).

Such a plan extension process can be performed iteratively, until the resulting structure is considered to be sufficiently robust to support flexible plan execution. The extended structure $\mathcal{P}_e$, resulting from such an iterative extension of the original partial order plan $\mathcal{P}$, bares similarity to both contingency plans [Warren, 1976; Peot and Smith, 1992] and probabilistic partial plans constructed by the BURIDAN planner [Kushmerick *et al.*, 1995]: On the one hand, as with contingency plans, $\mathcal{P}_e$ simultaneously captures alternative courses of actions. However, $\mathcal{P}_e$ has no schematic tree-like structure, with the tree nodes standing for the explicitly predefined branching points. On the other hand, as with probabilistic partial plans, a given precondition of an action in $\mathcal{P}_e$ can be supported (= potentially provided) by more than a single other action. However, the semantics of such an "extended support" in $\mathcal{P}_e$ is very different than in probabilistic planners such as BURIDAN. In what follows, we refer to such an extended structure as EPOPG.

Clearly, the constructed EPOPG $\mathcal{P}_e$ may not be a valid partial order plan, and this is actually the case with the EPOPG depicted in Figure 4(b). However:

1. It does contain at least one non-trivial, valid partial order plan, and
2. At every execution state s, the agent will still choose an action providing it with the maximal expected reward.

The first claim seems to be obvious, since any single action by itself is a valid partial order plan. However, the fact that (i) we start our planning process by constructing a valid plan for a set of capabilities, and (ii) nothing from this plan is ever removed during the "seed" extension, makes this statement more valuable.

The second claim is both less straightforward and very important, as maximizing expectation is the core part of our execution model. The soundness of this claim follows from the fact that, using the Refine procedure, the process of calculating $U(\mathcal{P}_e, s, \rho)$ via Eq. 3 exploits only valid total order sequences of actions from $\mathcal{P}_e$, even if $\mathcal{P}_e$ is not a valid partial order plan. The completeness follows from the fact that the first parameter of $U(\mathcal{P}, s, \rho)$ in Eq. 3 decreases monotonically with the nesting depth. We observe, therefore, that completeness is preserved even if the constructed EPOPG $\mathcal{P}_e$ contains cyclic dependencies[3].

4.2 Discussion

Our work leaves open a number of important issues for future research, and below we discuss some of them. Of particular importance is improving robustness of the planning stage and efficiency of the execution stage.

First, observe that the ability of an agent to adapt to new goals during the execution dramatically depends on the structure of the constructed EPOPG $\mathcal{P}_e$. More precisely, the actual contribution of extending the "seed" partial order plan depends directly on the strategy for choosing the plan extensions Υ. Clearly, various domain-dependent strategies can be developed, and their potential contribution is apparent. However, providing general, domain-independent principles that will rank the expected contribution of various extension strategies seems to be an important and interesting topic for the future research. In particular, results in this direction will contribute to the area of contingency planning. This is one of the issues we are currently examining, and here we informally discuss a general intuition that could be found helpful in pursuing this research direction.

Suppose that, after constructing the "seed" partial order plan $\mathcal{P}$ for capabilities, the agent is provided with the information that the value of one of its capabilities g is likely to be high during execution. (This covers both the initially assigned goals, and the initially irrelevant capabilities). In this case, it seems reasonable that the agent will strive to backup its plan $\mathcal{P}$ by inserting alternative *shortcuts* to the node g in the EPOPG being constructed. For instance, given the partial order plan in Figure 4(a), the action $\texttt{Navigate}(\texttt{L2}, \texttt{L3})$ added in EPOPG in Figure 4(b) can be seen as such a shortcut to the goal $\texttt{At}(\texttt{L3})$. Likewise, as we would like to control the size of the constructed EPOPG, the intention should be to use extensions Υ that provide shortcuts to as many such valuable goals as possible. However, as this seems to be reasonable at the level of intuition, in the future we would like to provide a clear formalization of this task in a domain-independent manner.

Second, consider the value estimation process that an agent performs at every decision point. We have already discussed that computing a precise value estimation is intractable for most, if not all, practically interesting domains. Therefore, we correctly began our discussion with an approximate estimation provided by Eq. 3, as this approximation procedure dramatically reduces the branching factor of the value iteration process. However, the complexity of calculating $U(\mathcal{P}_e, s, \rho)$ is still on the order of the number of alternative valid sequences of actions consistent with $\mathcal{P}_e$, s, and ρ, which in worst case is obviously exponential in the number of actions in $\mathcal{P}_e$. Therefore, to obtain truly practical execution schemes one will probably have to examine various techniques to limit the depth of value iteration with as little loss of decision accuracy as possible.

```
Loop forever:
  1. Check for and process new goal assignments $G \subseteq C$,
     where $C$ is the set of the agent's capabilities.
  2. Given the current state, generate a POPG $\mathcal{P}$ for the
     planning problem having all the capabilities $C$ of the
     agent as the goals $G'$ to achieve.
  3. Iteratively extend $\mathcal{P}$ to a EPOPG $\mathcal{P}_e$.
  4. For each capability $g \in G'$, annotate the corresponding
     nodes in $\mathcal{P}_e$ with $V_g$. In particular, if $g \notin G$, then
     $V_g = 0$.
  5. Loop until (all $g \in G$ are satisfied) $\vee$
     ($\mathsf{actions}(\mathcal{P}, s, \rho) = \emptyset$)
     (a) Estimate the value $U(\mathcal{P}_e, s, \rho)$ of the current ex-
         tended plan $\mathcal{P}_e$ using the value iteration process
         in Eq. 3.
     (b) Choose an action $A \in \mathsf{actions}(\mathcal{P}, s, \rho)$ that actu-
         ally provides $U(\mathcal{P}_e, s, \rho)$.
     (c) Perform $A$, resulting in the new state $\sigma(s, A)$ and
         some remaining amount of resource $\rho' \leq \rho$.
     (d) Set $\mathcal{P}_e = \mathsf{Refine}(\mathcal{P}_e, A, s)$.
     (e) Suspend all the goals $g$ that are no longer reach-
         able in $\mathcal{P}_e$.
     (f) Check for and process new goal assignments $\mathcal{G}$.
```

Figure 5: Planning and execution cycle of a single agent.

We believe that there are several ways to provide a good estimate of $U(\mathcal{P}_e, s, \rho)$, which in turn could be used instead of Eq. 3 starting from a certain depth of the value iteration process. Currently we are examining the method introduced by [Dearden *et al.*, 2002] for off-line backpropagation of the goal values to the internal nodes of a Graphplan-based plan graph, and in particular the applicability of this method to the non-leveled graphical structures such as POPG and EPOPG. Generally speaking, using this method, each action node A is associated with a value function $V_A(\rho, s)$ that provides an approximation of the combination of $\alpha(\mathcal{P}, A, s, \rho)$ and $\beta(A, \rho)$ from Eq. 2. This optional extension is not covered in this paper as it is still a topic of ongoing analysis.

5 Model of Opportunistic Execution

The model for plan generation and execution described in the previous sections provides an agent with enormous flexibility in selecting its course of action. This flexibility would in turn allow agents operating in real-world domains to better adapt to dynamic environments, especially in terms of the opportunistic satisfaction of suspended goals. Here we discuss the way our model supports opportunistic behavior of the agents.

Figure 5 summarizes our model for an agent's planning and execution cycle. Focusing on step 5e, we note that by taking a particular action A from the EPOPG $\mathcal{P}_e$, the agent may suspend one or more of its actual goals, namely those that are not achievable along all courses of action beginning with A. Normally, suspended goals will either be re-planned for in

[3]We omit the formal proof of this claim, as it can be easily derived from Eq. 3 and the Refine procedure. (Figures 4(c) and 4(d) provide an informal intuition behind the proof as they correspond to the result of Refine on EPOPG from Figures 4(b) and the actions $\texttt{Navigate}(\texttt{L1}, \texttt{L2})$ and $\texttt{Navigate}(\texttt{L2}, \texttt{L3})$, respectively.)

the next planning-execution cycle, returned to the task broker agent $\mathbb{B}$ for re-allocation to another agent, or abandoned completely.

During execution, however, conditions may change such that a suspended goal may indeed become achievable. An agent is said to exhibit *single-agent opportunism* if it can detect and respond to events and situations that may allow one of its suspended goals to be satisfied. The notion of single-agent opportunism has been widely discussed in the literature (e.g. see [Hammond, 1993; Pryor, 1996; Francis, 1995; Simina and Kolodner, 1991]). In our model, single-agent opportunism uses a form of *predictive encoding* [Patalano *et al.*, 1993], in which the agent examines the current plan to find other places where the suspended goal may be achieved.

In general, adapting a given plan to achieve additional goals can be computationally hard [Yang *et al.*, 1992]. However, in our model, an agent can examine and try to extend an EPOPG $\mathcal{P}_e$, the structure that is not required to represent a single valid plan. Thus, a predictive encoding of suspended goals for possible opportunistic execution can be performed much more efficiently than updating a plan that should remain consistent. More specifically, suppose that after performing an action A, the procedure Refine removes from the current EPOPG $\mathcal{P}_e$ an action A' because one (or more) of A's precondition nodes have also been removed from $\mathcal{P}_e$. Let us denote by $\mathcal{P}'_e$ the EPOPG resulting from the above refinement of $\mathcal{P}_e$. If A was necessary for achieving some assigned goal g, then g will have to be suspended. However, if there exists a POPG $\mathcal{P} \subset \mathcal{P}_e$ that achieves g, the agent could try to predictively re-encode such a sub-plan $\mathcal{P}$ into the refined EPOPG $\mathcal{P}'_e$ (using the EPOPG extension approach from Section 4.1), and to treat $\mathcal{P}$ as the chosen extension Υ.

For example, consider the EPOPG presented in Figure 2. Suppose that the agent performs the action $\mathtt{Navigate(L1, L2)}$ before executing $\mathtt{TakePicture(L1)}$. The goal $\mathtt{hp(L1)}$ would be suspended, because the sub-plan $\mathcal{P}' = \{\mathtt{TakePicture(L1)}\}$ has been pruned by the Refine procedure. The remaining plan could be extended by re-grounding $\mathcal{P}'$ at any other appearance of a node $\mathtt{At(L1)}$ in the EPOPG. The plan execution procedure would automatically reconsider opportunistically achieving $\mathtt{hp(L1)}$ if and when it encounters $\mathtt{TakePicture(L1)}$ again in the future. Unfortunately, in this particular example, there are no such places in our EPOPG, thus predictive encoding will be infeasible, and we need to seek alternative ways to achieve $\mathtt{hp(L1)}$.

As with single-agent opportunism, when the agents in a multi-agent system are capable of recognizing and responding to opportunities for each other's goals, we say the system exhibits *multi-agent opportunism*. In our model, when an agent suspends a goal g, it can notify the other agents in the multi-agent system that it cannot satisfy g. The agents receiving this notification can treat g as they would a newly assigned goal. That is, an agent $\mathbb{A}$ receiving a request to achieve g would first attempt to do so by opportunistically fitting g into its current plan. Unlike a newly assigned goal, however, if $\mathbb{A}$ cannot opportunistically satisfy g, it would not plan for it in the next planning cycle.

Suppose that $\mathbb{A}$ is currently acting according to an EPOPG $\mathcal{P}_e$, and g is reachable in $\mathcal{P}_e$ (i.e. $\mathbb{A}$ has not itself suspended and abandoned g). To opportunistically adopt a new goal g, $\mathbb{A}$ needs only to properly increase the value of g, V_g, in $\mathcal{P}_e$. While considering actions in the future, the execution module will implicitly adjust its intention with respect to this update. Note that even if $\mathbb{A}$ has not abandoned g, but has suspended it using predictive encoding for possible opportunistic achievement in the future, the node g will still be reachable in $\mathcal{P}_e$. Perhaps even more interesting, even if $\mathbb{A}$ is not itself capable of single-agent opportunism, it may still be able to provide opportunistic support for other agents, as long as the goals g suspended by these other agents are still reachable in the EPOPG of $\mathbb{A}$: The only thing that $\mathbb{A}$ has to do is update the corresponding value functions V_g in its EPOPG.

Finally, if g is not reachable in $\mathcal{P}_e$, and the agent $\mathbb{A}$ is capable of minimal re-planning, then $\mathbb{A}$ could create a new sub-plan $\mathcal{P}'$ just for achieving g starting from the current state. The EPOPG $\mathcal{P}_e$ could then be extended to include $\mathcal{P}'$, again allowing the plan execution procedure from Section 4.1 to decide upon the course of action to take.

6 Summary and Future Work

We have introduced a simple yet flexible model for planning and execution under uncertainty about resource consumption of the agent actions. This model is based on separating the qualitative and quantitative parts of the problem, and we see it as especially suitable for the systems where online re-planning by the agent as a way to revise its current plan to adapt to opportunities and/or failures in its multi-agent group cannot or should not considered. We have argued and illustrated that using this model in systems where little or no re-planning is possible, the agents can improve their own behavior, and assist other agents in the system. An additional aspect of the problem that we have studied in this paper is the practical complexity of extending single-agent opportunism to multi-agent opportunism. We have shown that our model arguably scales well and naturally supports multi-agent opportunistic behavior. In addition, and somewhat surprisingly, we have shown that as our model is devoted to minimal re-planning, assistance with the goals suspended by other agents can be more efficient than trying to find a way to still achieve the agent's own suspended goals.

Currently, we are implementing this model, planning to accomplish the implementation by August, 2003. Our intention is to test the approach on the problems extended from the standard benchmark domains used in the International Planning Competition (such as `Rovers` domain which we mentioned in the paper) [Fox and Long, 2002b]. Generating the "seed" POPG, as well as extending sub-plans for the constructed EPOPG, is based on an external off-the-shelf planner. Currently we are using the `LPG` planner [Gerevini and Serina, 2002], however any planner capable of producing plans for the domains described in the standard PDDL language [Fox and Long, 2002a] should be applicable.

We intend to conduct experiments in this environment to examine the performance impact of our plan execution model, especially when opportunities are being exploited by the agents. For these experiments, the broker agent $\mathbb{B}$ will randomly generate new goals and assign these goals to agents with the appropriate capabilities. Instead of assigning all of the goals at the beginning of a simulation run, however, the broker will generate batches of goals at regular intervals throughout the simulation, representing daily assignments for the various agents. Our experiments will evaluate the value obtained and computational costs incurred by the agents using a traditional sequential plan execution mechanism and our dynamic plan execution (both with and without opportunism).

Acknowledgments

The authors would like to acknowledge Prof Elise Turner and Prof Roy Turner at the University of Maine for their support of Mr. Lawton's contributions to this work.

References

[Blum and Furst, 1997] A. Blum and M. Furst. Fast planning through planning graph analysis. *Artificial Intelligence*, 90:281–300, 1997.

[Blythe, 1998] J. Blythe. *Planning under Uncertainty in Dynamic Domains*. PhD thesis, Carnegie Mellon University, 1998.

[Boutilier *et al.*, 1999] Craig Boutilier, Thomas Dean, and Steve Hanks. Decision-theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research*, 11:1–94, 1999.

[Bresina *et al.*, 2002] J. Bresina, R. Dearden, N. Meuleau, S. Ramakrishnan, D. Smith, and R. Washington. Planning under continuous time and resource uncertainty: A challenge for AI. In *Proceedings of the Eighteenth Conference on Uncertainty in Artificial Intelligence*, pages 77–84, 2002.

[Bylander, 1994] T. Bylander. The computational complexity of propositional STRIPS planning. *Artificial Intelligence*, 69(1-2):165–204, 1994.

[Dearden *et al.*, 2002] R. Dearden, N. Meuleau, S. Ramakrishnan, D. Smith, and R. Washington. Contingency planning for planetary rovers. In *Third International NASA Workshop on Planning & Scheduling for Space*, Houston, Texas, October 2002.

[desJardins *et al.*, 1999] M. desJardins, E. Durfee, C. Ortiz, and M. Wolverton. A survey of research in distributed, continual planning. *AI Magazine*, 4:13–22, 1999.

[Fox and Long, 2002a] M. Fox and D. Long. PDDL2.1: An extension to PDDL for expressing temporal planning domains, 2002. www.dur.ac.uk/d.p.long/pddl2.ps.gz.

[Fox and Long, 2002b] M. Fox and D. Long. The third international planning competition: Temporal and metric planning. In *Proceeding of the Sixth International Conference on AI Planning and Scheduling*, pages 333–335, 2002.

[Francis, 1995] A. Francis. *Memory-Based Opportunistic Reasoning*. Ph.d. thesis proposal, College of Computing, Georgia Institute of Technology, 1995. ftp.cc.gatech.edu/pub/ai/students/centaur/proposal.ps.Z.

[Geffner, 2002] H. Geffner. Perspectives on artificial intelligence planning. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence*, pages 1013–1023, 2002.

[Gerevini and Serina, 2002] A. Gerevini and I. Serina. LPG: A planner based on local search for planning graphs with action costs. In *Proceedings of the Sixth International Conference on AI Planning and Scheduling*, pages 13–22, 2002.

[Hammond, 1993] K. Hammond. Opportunistic memory. *The Journal of Machine Learning*, 10(3), March 1993.

[Klusch and Sycara, 2001] M. Klusch and K Sycara. Brokering and matchmaking for coordination of agent societies: A survey. In A.Omicini et al., editor, *Coordination of Internet Agents*. Springer Verlag, 2001.

[Kushmerick *et al.*, 1995] N. Kushmerick, S. Hanks, and D. S. Weld. An algorithm for probabilistic planning. *Artificial Intelligence*, 76(1-2):239–286, 1995.

[Lawton, 1999] J. Lawton. Opportunism in planning systems: A critical review. http://cdps.umcs.maine.edu/~jhlawton/opp-survey.ps.gz, 1999.

[McAllester and Rosenblitt, 1991] D. McAllester and D. Rosenblitt. Systematic nonlinear planning. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, pages 634–639, 1991.

[Multi-Rovers, 2002] Distributed Rovers Project, Planning and Scheduling Artificial Intelligence Group, JPL. www-aig.jpl.nasa.gov/public/planning/dist-rovers/, 2002.

[Ogston and Vassiliadis, 2001] V. Ogston and S. Vassiliadis. Matchmaking among minimal agents without a facilitator. In *Proceedings of the Fifth International Conference on Autonomous Agents*, pages 608–615, Montreal, Quebec, 2001.

[Patalano *et al.*, 1993] A. Patalano, C. Seifert, and K. Hammond. Predictive encodings: Planning for opportunities. In *Proceedings of the Fifteenth Conference of the Cognitive Science Society*, pages 800–805, 1993.

[Peot and Smith, 1992] M. A. Peot and D. E Smith. Conditional nonlinear planning. In *Proceedings of the First International Conference on AI Planning Systems*, pages 189–197, 1992.

[Pryor, 1996] L. Pryor. Opportunity recognition in complex environments. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence*, pages 1147–1152, 1996.

[Shehory and Kraus, 1996] O. Shehory and S. Kraus. Formation of overlapping coalitions for precedence-ordered task execution among autonomous agents. In *Proceedings of the Second International Conference on Multi-Agent Systems*, pages 330–337, 1996.

[Simina and Kolodner, 1991] M. Simina and J. Kolodner. Opportunistic reasoning: A design perspective. In *Pro-*

ceedings of the Seventeenth Conference of the Cognitive Science Society, pages 78–83, 1991.

[Smith *et al.*, 2000] D. Smith, J. Frank, and A. Jónsson. Bridging the gap between planning and scheduling. *Knowledge Engineering Review*, 15(1), 2000.

[Warren, 1976] D. Warren. Generating conditional plans and programs. In *Proceedings of AISB Summer Conference*, pages 344–354, University of Edinburg, 1976.

[Washington *et al.*, 1999] R. Washington, K. Golden, J. Bresina, D. Smith, C. Anderson, and T. Smith. Autonomous rovers for Mars exploration. In *In Proceedings of the 1999 IEEE Aerospace Conference*, 1999.

[Williams and Nayak, 1997] B. Williams and P. Nayak. A reactive planner for a model-based executive. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*, pages 1178–1185, Nagoya, Japan, August 1997.

[Yang *et al.*, 1992] Q. Yang, D.S. Nau, and J. Hendler. Merging separately generated plans with restricted interactions. *Computational Intelligence*, 8(2):648–676, 1992.

A Dynamic Environment Modelling Framework for Selective Attention

Thorsten Buchheim, Georg Kindermann, Reinhard Lafrenz, and Paul Levi
Institute of Parallel and Distributed Systems,
University of Stuttgart, Universit®sstr. 38, Dñ70569 Stuttgart
Phone: +49-711-7816-229, Fax: +49-711-7816-250
{buchheim,kindermann,lafrenz,levi}@informatik.uni-stuttgart.de

Keywords: Selective Attention, World Modeling, Communication, Multi-Agent-Systems

Abstract

Sensor data evaluation and environment modeling represent a central part in the design of robotic systems. The environment modeling process usually is done by several data processing modules by means of sensor data fusion, combination of a priori knowledge with currently sensed data, etc. Depending on the application scenario there is usually a trade-off between accuracy and time consumption of the environment modeling process. Some domains require a quite accurate environment model and processing time plays a secondary role. More reactive domains, however, demand for rather short cycle times, even if this results in a less accurate model. For some applications the required degree of accuracy may vary depending on the current situation or task to perform and some parts of the sensed information may be more important than others. Several architectures were proposed to address this problem by special mechanisms which schedule the available resources according to the needs of the current tasks which is usually denominated by the term ìselectiveî or ìfocusedî attention. In our work we will show how these concepts can be put into practice by a Ðexible and open software design which is applied for our *CoPs* RoboCup midsize team.

1 Introduction

Sensor data evaluation and environment modeling represent a central part in the design of robotic systems. To act and react properly on different situations an autonomous robot usually is equipped with various sensors to gather information about its surrounding which it uses to generate a model of its environment.

The environment modeling process usually is done by several data processing modules by means of sensor data fusion, combination of a priori knowledge with currently sensed data, etc. Most of the environment modeling approaches are structured in a hierarchical manner. While the lower evaluation levels usually extract simple features from the sensor raw data, like color blob detection in a 2D camera image or line extraction from a distance scan of a laser range Ýnder, the higher levels use these data to construct a more or less detailed map of the environment.

Most times the environment modeling framework is designed once for the application domain and the underlying sensor and actuator setup of the robotic system. However, there are application scenarios that require a high degree of Ðexibility within the environment modeling process regarding sensor equipment as well as the evaluation algorithms. Prior experiences with multi agent based software architectures have shown the usefulness of a high Ðexibility of multi sensor systems like for a multi agent based system for optical inspection [Buchheim *et al.*, 2000]. Here, different sensors and evaluation mechanisms were designed as separate agents which cooperatively solved the task of quality inspection of workpieces. A similar approach was also used for our Ýrst RoboCup team software design [Lafrenz *et al.*, 2000].

Another important aspect for autonomous systems is the time efÝciency of the environment modeling process. Depending on the application scenario, there is usually a trade off between accuracy and time consumption. While some domains require a quite accurate environment model and processing time plays a secondary role, e.g. in case of a robotic museum tour guide [Thrun, 1997], more reactive domains demand for rather time efÝcient algorithms.

According to the current situation or task to perform the required degree of accuracy may vary and some parts of the sensed information may be more important than others. Several architectures [Langley *et al.*, 1991; Tsotsos, 1997; Gat, 1991; Hayes-Roth, 1991] were already proposed to address this problem by special mechanisms which schedule the available resources according to the needs of the current tasks. These mechanisms are usually denominated by the term ìselectiveî or ìfocusedî attention.

Works like [Gat, 1991] and [Tsotsos, 1997] proposed extensions to Brookís subsumption architecture for attentive processing [Brooks, 1991]. In [Tsotsos, 1997] this is achieved by allowing behaviors not only to act in the physical world but also to manipulate its internal representation. A behavior is deÝned as a process which takes input in one or more representations and creates or manipulates other internal or external representations for subsequent processing. Gatís ATLANTIS architecture [Gat, 1991] deÝnes three layers for control, sequencing, and deliberation where the sequencing layer selects the stimulus-response mapping by activating and deactivating speciÝc modules in the control layer. The THEO architecture [Mitchell *et al.*, 1991] provides a means of selective attention by introducing the concept of eager and lazy sensing. While eagerly sensed features are updated constantly with each ìsense-decide-executeî control loop, lazily sensed features are deleted after the evaluation step and only re-sensed when explicitly needed.

All of these works motivate clearly the necessity for such mechanisms and present concepts for them. In our work we will focus on how to put these concepts into practice by a Ðexible and open software design which allows for an easy integration of all these methodologies.

We will introduce a Ðexible and modular framework for environment modeling which neither imposes restrictions on the type or amount of data to be stored nor the data processing or data generating elements (like sensors) and furthermore provides functionality to control the time cycles for data generation or processing depending on the current task or action of a robot. We then present a successful application of this framework for the RoboCup domain.

The paper is structured as follows: Section 2 presents the data management concept of the framework. Section 3 introduces the data acquisition and processing elements as well as the time cycle control mechanisms used for selective attention. The integration of this framework within the communication framework of the *CoPs* robotic soccer team [Lafrenz *et al.*, 2002] of the University of Stuttgart will be shown in Section 4. Section 5 then illustrates an application scenario of the complete framework within the *CoPs* -team. Section 6 concludes and gives an outlook to future applications and improvements of the framework.

2 Data Management

One of the central issues of the world modeling process is the representation of the data supplied by the various sensors and data processing modules of the system.

The focus of our world modeling design was to achieve a maximum degree of scalability and extendibility which allows for an easy integration of new sensory devices and evaluation algorithms. Secondly our work concentrated on performance issues like minimal memory consumption and least possible data volatility due to frequent memory allocation during run time. All of this is accomplished by a Ðexible data container concept which handles generic data objects of any arbitrary type via a common interface which will be described in the following.

2.1 Data Representation - Containers and Data Objects

In our concept *Data Containers* represent the key elements of data storage. They aggregate a number of so called *Data Objects* which hold generic (environment) data elements and provide administration functionality like adding, locating and accessing stored *Data Objects*.

The concept of data containers was Ýrst applied for a RoboCup simulation environment for the F2000 league [Kleiner and Buchheim, 2003] to facilitate the storage of generic data objects in a plug-in based architecture. Here it was also proposed as a uniform way of data representation for the development of generic sensor evaluation algorithms for RoboCup.

A *Data Container* stores all environment information of one robot which is represented by a set of different environment features. Each feature is stored within a separate *Data Object* and can vary in its degree of abstraction (e.g. sensory raw data like a simple camera image or top level information like an absolute localization in the environment).

The data objects are identiÝed by the following two attributes

1. the data source, the data stems from, e.g. a speciÝc sensor or a data processing module and
2. the data feature the object represents, e.g. a red color blob or a relative ball position.

The data representation itself is done by so called *Data Units (DU)* which form the core of the *Data Object*. A *Data Unit* stores the data itself in any arbitrary format along with its time stamp. Each *Data Object* maintains a history list of the n past *Data Units* over time (where n is a conÝgurable parameter). This data history can be essential for data evaluation purposes when e.g. data observations are made over time.

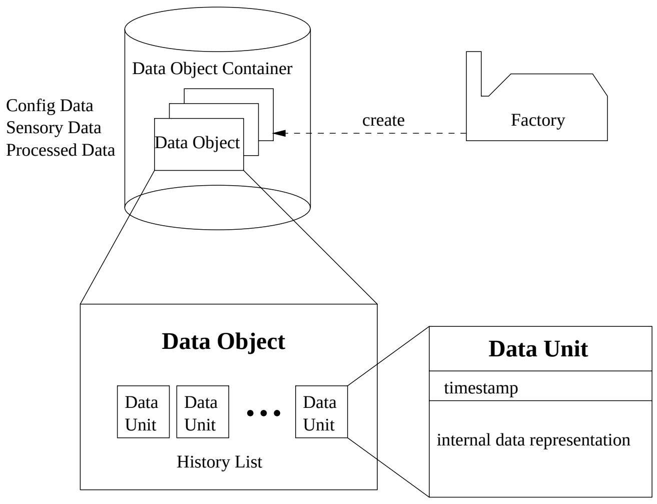


Figure 1: Data Container Concept

The creation of *Data Objects* is done according to the factory pattern [Gamma *et al.*, 1995]. Each new *Data Object* has to be registered with the factory by giving the two above mentioned data object identiÝers along with a data descriptor deÝning the data representation used for the *Data Unit* part of the *Data Objects*. Within the data descriptor the underlying basic data type and further information like the number of entries, data Ýelds per entry and history length are provided. After registration, the *Data Object* can easily be added to a *Data Container* just by providing its unique identiÝers.

Since most software approaches for autonomous robots use multiple threads to simultaneously process incoming data, much attention must be paid to mutual exclusion for shared data objects. These necessary data locking mechanisms sometimes bear negative side effects on the overall performance of a system, when e.g. a data processing algorithm locks data for an extended period of time while modules of another thread try to access it for reading.

For this reason *Data Objects* provide read and write slots to update or access their data and automatically implement the necessary data locking functionality. To simultaneously read and write data objects each *Data Object* contains besides its history list of *Data Units* one additional *Data Unit* as ìspare elementî. When requesting a write slot a pointer to this spare element is provided and this Data Unit is exclusively locked for the update process, while all other Data Units remain readable. When the update process is Ýnished the data slot is released and the spare element is inserted at the front of the history list, while the last element is extracted and becomes the new spare element.

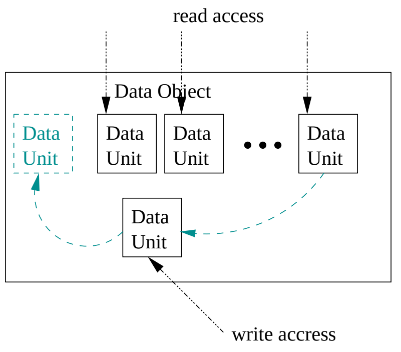


Figure 2: History Mechanism

In the CoPs team player architecture one *Data Container* is used to store the locally sensed and interpreted data of one robot which we call the robotís egocentric environment model. Currently data of the following categories are stored

- sensory (raw) data
- results of sensor data processing (sensor fusion and merging)
- control input, for RoboCup e.g. referee decisions
- control and conÝguration parameters affecting the behavior of some modules or the robot.

Usually, in a multi robot domain like RoboCup the robots partially share their data to either improve their view on the environment or to synchronize their actions. In our architecture we use a separate *Data Container* for each active robot transmitting data to store the communicated data. These communication *Data Containers* and the egocentric environment model form the complete world model of one robot of the *CoPs*-Team. Further details on the inter-robot-communication will be given in section 4.

3 Data Acquisition and Processing

The prerequisite for modeling the environment is the ability to sense it. Most robotic systems use multiple sensors to mutually compensate drawbacks of single sensors. The sensor data usually is processed in some way or fused with other data e.g. stemming from other sensors to obtain an accurate model of the environment. Both, sensors and processing algorithms should be easily integrateable into the system and should impose least possible restrictions for the developer. Within our framework *Sensors* and *Data Processors* are deÝned as so called *Information Sources* which manipulate *Data Objects* of the world model. *Data Processors* additionally require some input data to process which is retrieved from other *Data Objects*.

In the following, we will describe the sensor classes and the data processing units in detail and show the interplay with the world model.

3.1 Information Sources

Sensors and *Data Processors* represent the *Information Sources* of the framework which manipulate dedicated *Data Objects*.

Both, sensing and data processing are often parallelized by using multiple threads to avoid busy waiting for a sensor or an algorithm to provide new data. However, there may be situations where a Ýxed order of the incoming data is desired.

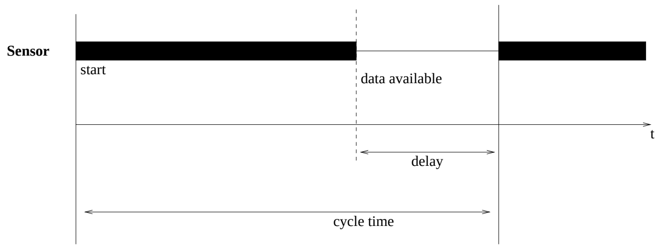


Figure 3: Sensor Cycle Time

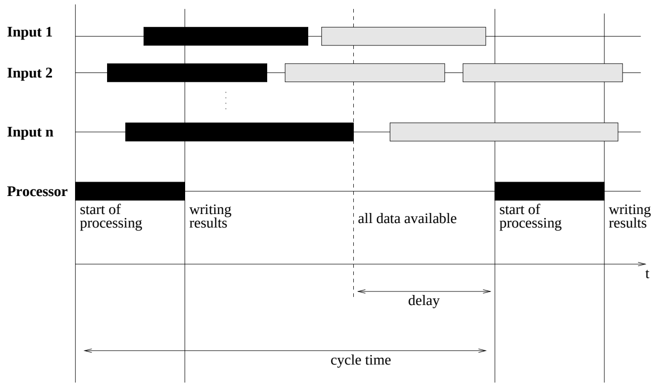


Figure 4: Data Processor Cycle Time

Our approach accounts for both requirements. For *Sensors* and *Data Processors* it is conÝgurable whether they are run in an individual thread or in a so-called ìcommon threadî, where they are processed sequentially according to a Ýxed order.

Since some of the components are very fast and others require a longer time to produce new data, we can conÝgure a cycle time for each thread. In opposite to cycle times in real-time operating systems, we donít have a *maximal* time consumption for each cycle but a *minimal* one, so we can avoid unnecessary checks, e.g. for a new video frame. If the frame rate is known, we are able to conÝgure an image analysis component with a cycle time of the frame rate. The cycle time mechanism is shown in Ýgures 3 and 4.

Attention Selection

By dynamically conÝguring the cycle times, it is possible to concentrate on speciÝc perceptions. This means for example that a driving robot can be conÝgured for short update cycles of the self-localization, whereas for a standing robot there is no need to constantly update its position once a reliable and accurate estimate has been found. The main advantage of this mechanism is that it can be used for attention selection. It is a very powerful tool for concentrating the available resources depending on the current situation.

Sensors

Sensors possess a pure data generation functionality. Sensor objects must automatically register their individual *Data Object Type* with the factory when they are created. During the registration process, additional parameters like the size of a single data element, the number of elements in a data set and the length of the history to store is given. With this information, the world model creates the required dedicated *Data Object* for the sensor which is then added to the *Data Container*. The *Data Object* can then be accessed via the *Data Container* by the name of the sensor and the feature type. The feature types themselves are independent of the underlying physical sensor, so that several sensors can provide similar features, e.g. lines relative to the robot in a Cartesian coordinate frame can either be provided by a camera or by a laser range Ýnder.

Data Processors

Data Processors can be integrated into the world model in the same way as *Sensors*. Each processor is connected in two ways to the world model: for reading and writing. It can read *Data Objects* of *Sensors* or other *Data Processors*. Then it writes the result of its own calculations into its own dedicated *Data Object* of the world model like the *Sensors*. That means, that data from other sensors are fused or transformed into information on a higher level of abstraction.

One example of a *Data Processor* could be a virtual line detection sensor, which yields the geometrical description of boundary lines in the environment. These objects can be e.g. lines drawn on the Ðoor, or walls. To detect these types of lines, one needs different physical principles of measurement. The lines on the Ðoor are reasonably detected by a vision system, whereas the walls are detected by a laser range Ýnder. Both information can be merged by a *Data Processor* implementing a so called ìvirtual line sensorî.

Another area of application is e.g. plausibility checking. In this case, the information of several different sensors can be used to Ýlter out wrong information. Often, complementary information is useful. A laser, for example, is not suitable to detect glass walls, whereas an ultrasonic sensor is. The combined information of both gives a more reliable model of the environment.

3.2 Data Driven Execution Cycles

The sensing mechanism follows a publisher-subscriber model [Gamma *et al.*, 1995]. In this context *Data Objects* act as publishers which *Data Processors* or in some cases *Sensors* can subscribe to. In case of an update of a *Data Object* all observing elements are automatically informed. Usually *Data Processors* subscribe to the *Data Objects* they need for their data processing. This is useful if data processing shall only be done when all input data is available or when the data Ðow of data processing mechanisms is to be monitored. *Sensors* usually only register for *Data Objects* containing conÝguration information to facilitate changes of the conÝguration at runtime.

Figure 5 shows an exemplary collaboration of *Sensors*, *Data Objects* and *Data Processors*. The publisher subscriber relation is marked by the dot ended lines. The Ýlled and empty dots show the notiÝcation state of each *Data Processor* for the corresponding *Data Object*. In this example it is assumed that each *Data Processor* only initiates its processing step when all *Data Objects* are updated, thus yielding a data driven execution cycle.

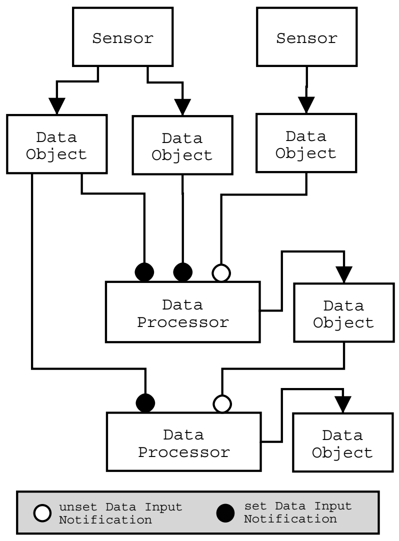


Figure 5: Data driven execution

One example application is the self-localization by Markov or Monte Carlo methods. Here, different types of sensor features are used and the likelihood of being at a speciÝc place is calculated dependent on the sensory information. A Monte Carlo *Data Processor* thus registers with all relevant *Data Objects* and uses this information to update its dedicated *Data Object* containing the absolute position.

3.3 Configuration

Each *Data Processor* and *Data Container* possesses an individual ConÝguration-*Data Object* which it is automatically subscribed to. As one essential part this conÝguration contains the already mentioned conÝgurable minimal cycle time.

With this mechanism it is possible to tune the system so that the sensors are evaluated at a time where the new data is available. In case of a camera sensor, the minimal cycle time should be at least the video frame rate. In an ideal case, each sensor runs in a separate thread at approximately its hardware cycle time. In this case, this does not only assure the most recent available information but also a constant update rate (ìheart-beatî) for each sensor. This is especially useful for time-dependent calculations, for instance Kalman Ýltering for optimal state estimation. For the attention selection we can conÝgure this heart-beat dynamically depending of the current role that is currently assigned to the robot [Lafrenz *et al.*, 2000].

4 Communication

For team communication in our *CoPs* architecture we use a publish/subscribe event channel approach [Buschmann, 1996]. The advantage of this concept is that the supplier and the consumer of information do not need to know each other (decoupling) and only the used channels have to be known. The communication framework allows to connect to one or more event channel(s) as supplier (publisher) and/or consumer (subscriber). This allows to implement different levels of communication like differently prioritized or thematically separated channels.

4.1 Communication Levels

Our implementation uses a CORBA event channel [OMG, 1995] using the push model which means that a supplier pushes its information to a channel. The channel is realized by a server process which runs on one of the robots or on an external computer which broadcasts the provided information to the subscribers.

The modules pushing the data to the channels can be implemented within an own conÝgurable thread or a common thread which is the same mechanism we already described in section 3.1. The common thread is normally used for triggering the transmission of *Data Objects* whenever they change, while individual threads are normally used to provide a constant ìstreamî of information with a conÝgurable cycle time.

At the moment we use three communication levels, according to the known reactive, tactical, strategic level approach:

1. Information acquired from the local sensors
2. Derived data from local view and the received data from level one
3. Data processed by observing the local view and the information of level 1 and 2 over an extended period of time.

This enumeration also reÐects the frequency of data exchange. In our application each robot provides a snapshot of its local view approximately every $250ms$ and is implemented by an own thread (level 1). Pushing the data of the other levels is implemented within the common thread and so dependent on how often the information is updated. Information of level 2 usually is exchanged about every $5 - 10s$ while information of level 3 is normally transmitted about every half a minute or more.

Level one data is intended to be used to extend the local view derived from the information of the own sensors. Each robot makes its own decision about what action he should execute next, based upon its knowledge of the *real world*. Restricted to the local view of a robot cooperative playing is a difÝcult task which can be considerably reduced in its complexity by the use of communication. For example a decision which robot of a team should approach the ball would require that each player performs a complete analysis of the current situation concerning the positions of all team mates as well as the ball to determine the player next to the ball. With each robot exchanging its own current distance to the ball with its team mates a solution can easily be found.

Level two data should be information which changes less frequently, e.g. the role of a robot (defender, attacker) or an estimation of the situation. Another point is providing additional information for other robots. For example a defender is often off duty while the ball is near the opponent goal. So our concept allows to change its attention by reducing the processing time for normal situation analysis and investing it for other purposes. So more sophisticated, time consuming analysis on its own sensor and/or the communicated information can be done. For example the robot can provide a (rough) estimation about the absolute position of the attacking robot, which is normally crowded by opponent robots and so not able to localize itself. This can be done for example by considering the communicated relative ball position of that robot and the own detected absolute ball position.

Level three could for example be used for negotiations and conÐict resolutions. Considering the defender scenario mentioned above, this defender could detect some inconsistency, for example an incorrect localization of the attacker due to a sensor malfunction. At the moment this level is mostly used to signal possible problems to the human coach.

4.2 Communication Data Handling

Information received from other robots is stored in one *Data Container* for each robot. If a new robot enters the game, it registers itself at the different channels. So if a robot pushes its information to a channel for the Ýrst time the registered consumers receive this information. For storing the information of the new robot the others create a new instance of a *Data Container* and add the name of the new robot to their list of active robots which is stored in a *Data Object* within their local *Data Container*. This local *Data Container* builds the egocentric environment model of a robot (see section 2.1). It stores the information acquired from the local sensors and data processing results from communicated or sensory data. The information received from other robots is separately stored in so called *Communication Data Containers*. The *Data Objects* within this container are only updated by the communication module and read by *Data Processors* to compute information which is stored within the local, egocentric *Data Container* (see Ýgure 6).

The software design allows for an easy implementation and integration of new *Data Processors* for the communicated information. The features of the implemented publisher/subscriber model we described for the *Data Objects* in section 3.2, are also provided for the *Communication Data Objects*. By using these mechanisms it is easily possible to implement *Data Processors* which are initiated each time a

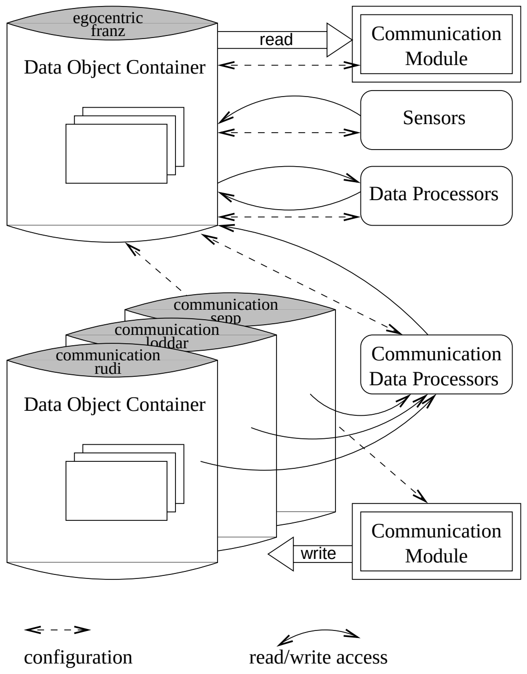


Figure 6: Communication Containers

robot sends a speciÝc information or just when this information item is received from all active robots.

For administrative reasons, e.g. to detect that a robot dropped off, a *Data Processor*, the *active robot observer (ARO)* is started to observe the list of active robots and the containers holding the appropriate communication data. It is expected, that information of each robot is received regularly. So if no information is received from a robot for some time period, this robot is viewed as not active anymore and removed from the local list of active robots. Additionally another datum is updated, which is registered to be transmitted (normally level 3), to inform the other robots and the human coach about the possible drop off. Normally all *Data Processors* for communicated data are registered with the *Data Object* holding the list of active robots. So if the ARO removes the robot from the list, they are informed and can initiate an appropriate reaction.

4.3 Selective Attention for Communication

Selective attention can be achieved on the supplier as well as on the receiver side. For the supplier it is possible to conÝgure the *Data Processors* producing the data to be transmitted and the communication modules (e.g. thread cycle times) responsible for sending it. This way it is possible to adjust the amount, quality and focus of information it provides depending on the current situation.

On the receiver side, it is possible to selectively choose which data from which robot or group of robots should be examined or ignored, by appropriately conÝguring the responsible *Data Processors*. So it is possible to control how much attention the receiver pays to which information provided by different robots.

For example the attacking robot can normally ignore information from other robots, which is of low plausibility and importance, while a defending robot, can do additional plausibility checks on the provided information, try to detect problems and initiate a conÐict resolution procedure if necessary.

5 Selective Attention in the *CoPs* robotic soccer midsize team

Robotic soccer is a domain which usually requires both, fast algorithms for quick reactive behavior like dribbling or intercepting a ball and a high accuracy which is needed for strategic team behaviors, like e.g. defending the own goal area or for strategic positioning. Fortunately, most situations do not require both qualities at the same time which is the point where selective attention comes into play.

In our current approach of selective attention in the *CoPs-Team* we control the activity of the *Data Processors* of the environment model in two ways. Firstly by manipulating individual cycle times as described in section 3 and secondly by deÝning a special data input for explicitly triggering the execution of a particular *Data Processor*. This special ìtrigger inputî is used like any other regular data input and has to be set to initiate an execution cycle. This allows to trigger special data processing routines ìon demandî.

Currently we use a role based control of selective attention for our team. In our team architecture there are three top level roles a player can be in:

- The *active player* is the one who usually goes for the ball or currently possesses it and tries to mark a goal.
- The *supporter* assists the active player by following him to catch the ball if the active player fails in executing his current action (e.g. rebound shot).
- The *defender* defends the goal the best way possible according to the current situation.

Here, the active player requires the highest reactivity to get or control the ball while the defender usually can use its resources to obtain a more accurate model of the environment. The roles are switched dynamically which means that when an attack of the opponent team occurs the defender automatically becomes the active player when the ball gets into the vicinity of the own goal.

Each robot player of the *CoPs* team is currently equipped with the following sensors:

- *SICK* laser range Ýnder (LRF) providing distance data in a $180°$ angular Ýeld of view with a $1°$ resolution which is directly fed to a line and obstacle detection algorithm
- 2D camera detecting color blobs of the ball and goal colors
- a ring of 16 sonar sensor modules each providing one single distance measurement within their sensing range
- an odometry sensor of the Nomadic Scout robot platform, providing a position (x, y, θ) within an absolute odometry coordinate frame which, however, is biased by slippage of the wheels during the motion of the robot. Additionally the current wheel speeds are measured by taking the momentarily applied motor currents.

The LRF and the camera are designed each as *Sensor* threads with controllable cycle times. The odometry data and the ultrasonic sensors are updated each time the scout hardware platform is accessed when e.g. setting the wheel speeds of the robot.

Several data processing units process the incoming data and provide some meta level information on the environment or auxiliary information to support the decision making process for player actions which is done within a separate action execution layer which will not be described any further for the sake of simplicity. Some important sensor data processing elements are:

- Transformation of relative data into the odometry coordinate frame
- Kalman Ýltering of ball data
- Object tracking of laser detected objects
- Estimation of an outer bound of action
- Path Planning for obstacle avoidance
- Monte Carlo Localization.

A central problem within the midsize league is to determine the absolute position within the Ýeld. Besides the obvious advantage of absolute positioning for strategic team play, there is a further important aspect of a Ýxed coordinate frame which allows a motion-invariant representation of sensed data. Many algorithms like object tracking rely on the fact that the data is provided in a stable coordinate frame as given by a static observer. This data is then used to derive some motion information by matching successive measurements over time.

Since a reliable and fault tolerant localization mechanism cannot be assured at all times, we recurred to the absolute coordinate frame of the odometry sensor. Although this coordinate frame slightly shifts and rotates over time due to the slippage of the wheels, it turned out to be sufÝciently stable for the limited time frame in which the object tracking in RoboCup usually is done.

One processing element is thus concerned with the transformation of incoming sensor data from the camera and the LRF to the odometry coordinate frame. One *Data Processor* applies Kalman Ýltering on these data to stabilize the ball position estimate and calculate its motion vector, while another processor does the same for obstacles detected by the laser.

Path Planning is used when a direct way to an intended target position is obstructed by obstacles. In the absence of an absolute localization, it is useful to deÝne an outer bound for the path planning process, to avoid planning outside the Ýeld borders. To achieve this a *Data Processor* buffers detected lines from the laser within the odometry coordinate frame over a dedicated time frame which is then used for a guess of an outer bound where planning should take place.

We chose to implement the path planning as a separate *Data Processor* instead of integrating it in the action execution layer. The path planner is triggered on demand by the action execution layer, when needed to navigate around obstacles to reach a given target position. After having planned an appropriate path, the path planning *Data Processor* constantly checks whether the current path is still valid or if a replanning has to be done due to the motion of certain obstacles.

Although all these approaches work nicely, effective team play demands for an absolute localization. With an absolute position within the Ýeld the robots can share this information to organize their behavior by e.g. choosing appropriate positions within the Ýeld or assigning non-conÐicting roles to each player. For the absolute localization we use a Monte Carlo approach which uses the detected lines of the laser and the color blob data of the camera to estimate an absolute position in the Ýeld. This algorithm, however, has the drawback to be rather time consuming depending on the number of features used for localization and the intended accuracy of the position estimate. Hence, localization is done as seldom as possible. For the rest of the time a reliable estimate is propagated forth by the odometry updates and Kalman Ýltering.

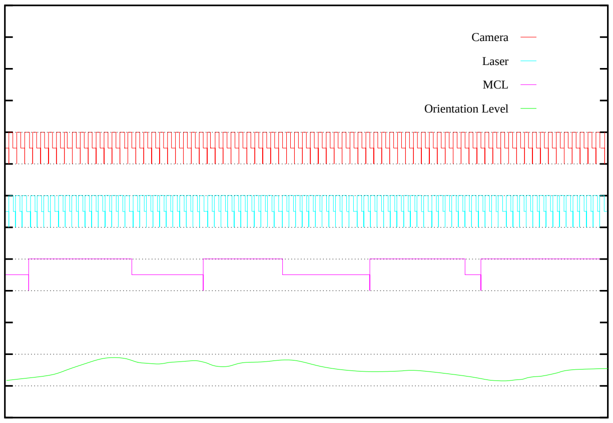


Figure 7: *Sensor and processing activity for active player role*

In our approach we partially renounce on a good absolute positioning for the active player in favor of a higher reactivity while the defender and supporter focus on determining an exact position within the Ýeld. However, there are situations when the degree of information for a current action falls below a critical limit and leads to a state of disorientation, like the occlusion of the opponent goal when dribbling the ball and a missing absolute localization within the Ýeld.

In these cases some resources have to be spent for the localization at the expense of a temporary lower reactivity. For the more passive roles, on the contrary, once a good position estimate has been found the activity of other *Data Processors* can be intensiÝed like ball or opponent tracking.

To account for this we deÝne a quality measure for the current state of the self localization (*orientation level*) which is used for determining the cycle times of the various *Data Processors*. This quality measure is derived from the Monte Carlo localization by the given probability of the best position estimate, its variance, and by a distance measure to other concurrent position hypotheses. Each role deÝnes a minimum value for this localization measure which should be maintained during its execution time. The selective attention control mechanism then tries to appropriately control the cycle times of the various *Data Processors* to assure the needed level of orientation.

Figure 7 shows the activity of the camera and laser sensor together with the Monte Carlo Localization (MCL) *Data Processor* and the current level of orientation while the robot dribbles the ball. Each *Data Processor* in the diagram has three states, inactive (zero level), waiting (0.5 level) and busy (1 level). The diagram clearly shows a lower activity of the MCL *Data Processor* while the level of orientation remains above a certain level. This activity increases when the state of orientation gets critical by shortening the waiting period. On the other hand the laser and camera update cycles are relatively short to achieve a high reactivity.

6 Conclusion and Outlook

In this work we presented a Ðexible and generic software framework for world modeling which enables an easy realization of selective attention mechanisms. Until now we merely focused on the software realization to facilitate these mechanisms and applied them for our RoboCup midsize team architecture. Currently the selective attention modeling is done manually based on the current action and situation of a particular robot. For the future we intend to integrate learning mechanisms for attention control which automatically determine which features of the world model are most relevant in certain situation. In this context we will especially focus on cooperative sensing to cooperatively build a consistent and accurate world model by means of communication and load balancing among team mates.

References

[Brooks, 1991] R. A. Brooks. Intelligence without representation. *Artificial Intelligence Journal (47)*, pages 139ñ159, 1991.

[Buchheim *et al.*, 2000] T. Buchheim, G. Hetzel, G. Kindermann, and P. Levi. A multi-agent approach for optical inspection technology. In *The Sixth Pacific Rim International Conference on Artificial Intelligence (PRICAI)*, 2000.

[Buschmann, 1996] Frank Buschmann. *Pattern-Oriented Software Architecture. A System of Patterns*. John Wiley & Sons, 1996.

[Gamma *et al.*, 1995] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns*. Addison Wesley, 1995.

[Gat, 1991] E. Gat. Integrating planning and reacting in a heterogeneous asynchronous architecture for mobile robots. In *SIGART Bulletin 2*, pages 70ñ71, 1991.

[Hayes-Roth, 1991] B. Hayes-Roth. Architectures for intelligence. In *SIGART Bulletin 2*, pages 301ñ321, 1991.

[Kleiner and Buchheim, 2003] A. Kleiner and T. Buchheim. A plugin based architecture for simulation in the f2000 league. submitted to RoboCup Intl. Symposium, Padua, 2003.

[Lafrenz *et al.*, 2000] R. Lafrenz, N. Oswald, M. Schule, and P. Levi. A cooperative architecture to control multi-agent based robots. In *The Sixth Pacific Rim International Conference on Artificial Intelligence (PRICAI)*, 2000.

[Lafrenz *et al.*, 2002] R. Lafrenz, M. Becht, T. Buchheim, P. Burger, G. Hetzel, G. Kindermann, M. Schanz, M. Schul¥, and P. Levi. Cops-team description. In A. Birk, S. Coradeschi, and S. Tadokoro, editors, *RoboCup-01: Robot Soccer World Cup V*, pages S. 616 ñ 619. Springer Verlag, 2002.

[Langley *et al.*, 1991] P. Langley, K.B. McKusick, J.A. Allen, W.F. Iba, and K. Thompson. A design for the icarus architecture. In *SIGART Bulletin 2*, pages 104ñ109, 1991.

[Mitchell *et al.*, 1991] T.M. Mitchell, J. Allen, Chalasani P., J. Cheng, O. Etzioni, M. Ringuette, and J.C. Schlimmer. Theo: A framework for self-improving systems. In *Architectures for Intelligence*, pages 325ñ355, 1991.

[OMG, 1995] Object Management Group OMG. CORBA services: Common Object Services SpeciÝcataion. OMG Document Number 95-3-31, 1995.

[Thrun, 1997] S. Thrun. The museum tourguide project: Experiences with a deployed service robot. In *Proceedings of the IEEE International Symposium on Computational Intelligence in Robotics and Automation*, 1997.

[Tsotsos, 1997] J.K. Tsotsos. Intelligent control for perceptually attentive agents: The s* proposal. In *Robotics and Autonomous Systems*, pages 5ñ21, 1997.

Intelligent Execution Monitoring in Dynamic Environments

Matthias Fichtner **Axel Großmann** **Michael Thielscher**

Department of Computer Science
Technische Universität Dresden
Dresden, Germany

Abstract

We present a robot control system for known structured environments that integrates robust reactive control with reasoning-based execution monitoring, allowing for action failures due to the interaction with humans and dynamic changes in the environment. On the reactive level, the robot is controlled using a hierarchy of low-level behaviours. On the high level, a logical representation of the world enables the robot to plan action sequences and to reason about the state of the world. If the execution of an action fails, high-level reasoning allows to infer possible explanations and to recover from the failure situation. The logic-based world model takes into account temporal information about changes in the environment. It is updated both using information from the sensory processing modules as well as by executing sensing actions. The proposed system is evaluated in several realistic office-delivery scenarios.

1 Introduction

In recent years, robotics has been subject to promising advances in sensor and actuator hardware, sensory processing techniques, and low-level control methods. Yet, if we want a mobile robot to perform complex tasks in real-world environments, we still face a number of problems. The information the robot has about its operating environment might be out of date, incomplete, and uncertain. The execution of actions might fail due to a multitude of reasons. Ideally, we want the robot to reason about unexpected situations and to infer possible explanations in order to recover from the failure situation.

The capabilities described above require the robot to maintain information on its own state, usually obtained by processing the sensory data, as well as knowledge about the operating environment and the task at hand, commonly referred to as world model. To deal with the uncertainty in the robot's observations, it is common practise to represent state information such as the robot's location in the environment or the position of objects of interest using state enumeration and probabilities. Popular approaches include position probability grids [Fox *et al.*, 1999] and particle sets [Thrun *et al.*, 2000]. On the other hand, as the robotic tasks become more complex, we would like to use reasoning and planning techniques. These require a logical representation instead. There is still a large gap between state-of-the-art reactive control mechanisms using probabilistic representations and symbolic reasoning and action planning methods for mobile robots. This is partly due to unrealistic assumptions in cognitive robotics approaches.

In order to make efficient use of high-level control for task planning and error recovery in dynamic real-time environments, temporal information about changes in the environment need to be incorporated into the world model. Therefore, we decided to investigate the following two kinds of data acquisition. The robot may obtain information about the environment using its sensory processing system concurrently while executing a sequence of actions or it may choose to execute specific sensory actions.

The aim of our work is to enable a mobile robot to perform action planning and reasoning-based execution monitoring in known structured environments, allowing for action failures due to the interaction with humans and dynamic changes in the environment. We present a hierarchical, modular control architecture that integrates low-level behaviours with a high-level controller, thus combining the robustness of reactive control with the power of intelligent reasoning. We have developed a layered scheme of execution monitoring. It allows the robot to find explanations for action failure using the logic-based world model and the history of the task execution. The capabilities of this novel kind of execution monitoring are evaluated using realistic scenarios for an office-delivery robot.

The paper is organised as follows. In Section 2, we discuss related work on logic-based representations of dynamic information and on execution monitoring. In Section 3, we describe the main components of the proposed architecture and the representations and techniques used at the reactive and the abstract level. In Section 4, we describe the concepts and techniques for representing dynamic information, the acquisition of information while executing a plan, and active sensing. In Section 5, we demonstrate the functionality of the system using example scenarios. We conclude in Section 6.

2 Related Work

There is a vast literature on the integration of planning and reactivity in autonomous mobile robots, e.g., on the traditional sense-plan-act architectures as well as behaviour-based control and variations thereof [Gat, 1992; Saffiotti, 1993]. Several of the issues related to that are discussed in the following sections. However, a detailed discussion of the previous work on this topic goes beyond the scope of this paper. In this section, we focus on literature on the realisation of reasoning-based execution monitoring and on logic-based representations of dynamic worlds instead.

2.1 Execution Monitoring

The problem of execution monitoring can be addressed in various ways. The individual solutions usually depend on the control architectures and the tasks they are used with. For example, there are efficient methods for handling errors in navigation tasks, e.g., [Stentz, 1995]. Moreover, there is a large body of related work in the fields of fault detection and isolation (FDI) and industrial control. Since we want our solution to be applicable to complex delivery tasks, we address the problem of execution monitoring in a logical framework, going far beyond the requirements of pure navigation tasks.

There is no generally accepted definition of execution monitoring. [Giacomo *et al.*, 1998] defined execution monitoring as 'the robot's process of observing the world for discrepancies between the actual world and the robot's internal representation of it, and recovering from such discrepancies'. In this work, we extend this notion in the way that the robot should come up with explanations for the detected discrepancies as well. Consequently, we can divide the overall process of execution monitoring into three steps: detecting discrepancies, explaining the situation, and launching a recovery procedure. They are discussed in the remainder of this section.

Detecting Discrepancies

The execution of complex actions generally requires the robot to have a representation of its current state – such as the robot's position in the environment, the distance to obstacles, and the state of the gripper – as well as a model of the environment and a description of the task to be solved. By comparing the current state with the model of the world and the task, it should be possible to detect erroneous situations. However, we do not expect such a comparison to be straightforward as the models and representations are likely to be complex and incompatible. In general, the representation of the robot's state will be layered and distributed. The control architectures usually include specialised modules for sensor data processing. At the low levels of control, we mostly use probabilistic representations. At the highest level of control, in contrast, we prefer symbolic, logic-based representations.

Suppose the robot is to execute a sequence of actions. At the beginning, we usually have an expectation of the intended effects, i.e., the change of state caused by each action. These expectations are going to be layered and distributed as well. The anticipated effect of a go-to action at the low level, for example, is a change of the robot's position within a certain amount of time while maintaining a minimum distance to obstacles. At the highest level of control, expectations can be inferred using a knowledge base and an underlying action theory.

Providing Explanations

Once an action had not the intended effect, we would like to know the reason, i.e., find an explanation for the encountered discrepancy between state information and expectation. In general, this will require reasoning. This goes beyond the capabilities of reactive control and has to be performed at the highest level of control.

The robot can generally only observe symptoms of the current situation. For instance, a robot getting stuck could have been caused by a variety of reasons: a localisation failure, a visible obstacle (a person), an invisible obstacle (a chair leg), an unexpected change of the environment (a door being closed), etc. On the other hand, the more relevant features of the environment are included in the state information and the smaller the granularity of the world model and task description, the easier it is to make conjectures about the possible reasons of failure.

Hierarchical planning meets the requirements mentioned above. If, for example, a complex navigation task is broken down into smaller actions like door passings, corridor and room traverses, then failures can be substantiated with higher reliability. By using default logic [Reiter, 1980], the planning and reasoning module can abstract away from the sheer non-exhausting, but increasingly unlikely, set of preconditions, thus solving the qualification problem. In case of unexpected situations though, these default assumptions must be double checked according to an ordered preference list [Martin and Thielscher, 2001], thus providing the most likely explanation for action failure.

Recovering

Once an explanation of the current situation is found and the state information and world model are corrected, some recovery strategy is expected to remedy the failure. In approaches such as [Fernández and Simmons, 1998], it is suggested to launch a predefined recovery plan. For example, when the robot notices that it ran into a dead end, it computes a path that brings it back on the original track and continues with the initial plan. Given a strong planning tool, instead of just correcting the mistake, we are able to find an optimal plan for the current situation (provided that the state and world model are correct). Suppose the robot did not run into a dead end, but found a shortcut. Now, the planner can provide a better solution if returning to the old track proves to be more costly.

2.2 Logic-based Representations of Dynamic Worlds

Previous logic-based representations of dynamic worlds are rooted in the general framework developed in [Sandewall, 1989], in which dynamic information is modelled by autonomous processes that run in parallel and that may eventually trigger further changes in the environment. This technique has been integrated, for example, into situation calculus [Reiter, 1996], event calculus [Shanahan, 1990], or fluent calculus [Thielscher, 2001a]. Implementing these approaches, the agent programming and planning languages ConGolog [Giacomo and Levesque, 2000] or FLUX [Martin, 2003] support the specification of concurrent processes and their effects.

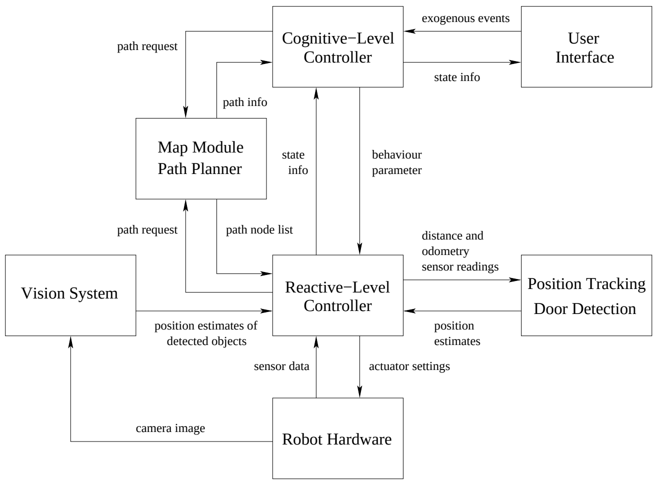


Figure 1: Control architecture of the robot.

A disadvantage of these methods is that all changes need to be explicitly inferred as effects of endogenous events, whose occurrence in turn needs to be derived from the ongoing processes. Robots which follow this approach in highly dynamic environments would be overwhelmed with constantly calculating all changes that happen around them. Instead, we represent temporal knowledge about a dynamic property by attaching the time of its observation to the corresponding property. Decay of information will then be simulated by automatically forgetting about outdated fluents after a certain amount of time so that only recent knowledge is considered when devising plans.

3 Building the System

To put the functionality described above into practise, we added a high-level planning and reasoning component to a fairly standard hierarchical robot control system. In the following, we describe the parts of the system that are relevant specifically to execution monitoring.

3.1 System Architecture

The control architecture of the robot, as depicted in Figure 1, consists of several modules. The hardware controller talking directly to the robot's sensors (odometry, sonars, laser) and actuators (drive motors, gripper) is considered the lowest level of control. The basic perceptual and behavioural functions of the robot are implemented by the reactive-level controller. We have used a behaviour-based approach. That is, the reactive controller includes several interacting, task-specific programs that are referred to as low-level behaviours. Each behaviour program takes the current sensor readings and the state information and computes target values of the robot's actuators. Individual behaviours can overwrite the output of other behaviours. There are specialised sensor-processing modules for visual object detection, laser-based position tracking, and door detection. These components maintain a probabilistic representation of the detected objects, robot poses, and door angles, respectively.

The robot's goal-oriented behaviour is directed by the cognitive-level controller. This is done by setting task specific parameters of the low-level behaviours such as the activation context and target coordinates. To navigate in the office environment, both the reactive and the cognitive controller obtains information from the map and path planning module.

3.2 At the Lower Levels

Independent of the task to be performed, the safety of the robot has to be maintained at all times. Therefore, the reactive controller includes a set of low-level safety behaviours, e.g., for obstacle avoidance and velocity control, that cannot be switched off by higher levels of control. The other low-level behaviours are designed to achieve specific (parameterised) goals such as to travel to a target position or to pick up an object. Suppose the robot is to execute a sequence of high-level actions. Then for each action, there is a designated process that supervises the execution of that action. This execution monitoring process invokes the appropriate low-level behaviours and deals with exceptional situations. In the following, we illustrate this concept for the action of travelling to a given office.

The monitoring processes are implemented as finite state machines. Some states are common to all actions, others are specific to the task. In the example, we have used the following states:

Init	Initialise the process
Deactivated	Wait for activation
WaitForGoal	Wait for target from cognitive controller
RequestPlan	Query the path planner
ReceivePlan	Receive path node list
PlanReceived	Activate low-level behaviour *GoToPos*
ExecutePlan	Set new intermediate target position and react on state info from *GoToPos*

For each monitoring process, there is a predefined set of exceptions, represented by state information. There are exceptions that are passed on by the low-level behaviours and there are exceptions that were detected by the sensor processing systems. The low-level behaviour *GoToPos* was implemented as finite state machine, too. The common interface to low-level behaviours consists of three states: *Init*, *Deactivated*, and *Running*. The state information used in the example above are:

InProgress	Execution in progress
Success	Execution terminated successfully
FailureStalled	Drive motors stalled
FailureObstacle	Path blocked by obstacle
FailureDoor	Path blocked by door
Timeout	Execution timeout
Interrupt	Execution interrupted

The monitoring process passes this state information on to the cognitive controller.

3.3 At the Highest Level

The high-level controller maintains a symbolic world model. Reasoning about actions is used at this level to plan complex tasks and to generate expectations as to the effects of actions. When a discrepancy arises between the expectations and the actual situation, the high-level control uses its reasoning facilities to come up with suitable explanations and a recovery plan. As the underlying action theory we use the fluent calculus with its solution to the classical frame, ramification, and qualification problems. Our system builds on the inference engine FLUX for the fluent calculus [Thielscher, 2002].

Specifying Actions

The cognitive controller requires precondition and effect specifications of each high-level action. To account for unexpected action failure, we make the distinction between normal and abnormal preconditions. The former need to be ascertained before an action can be planned while the latter are assumed away by default but serve as possible explanations in case the action surprisingly fails. For example, the following axiom specifies the preconditions of the action *Deliver*(o,p) of delivering object o to person p:

$$\begin{aligned}&\textit{Poss}(\textit{Deliver}(o,p),s) \equiv\\ &\quad \textit{Holds}(\textit{Carries}(o,p),s) \wedge\\ &\quad \exists r\ \textit{Holds}(\textit{InRoom}(r),s) \wedge \textit{Office}(r,p) \wedge\\ &\quad \neg \mathrm{Ab}(\textit{Traceable}(p),s) \wedge \neg \mathrm{Ab}(\textit{NotLost}(o),s)\end{aligned} \tag{1}$$

Here, the standard predicates *Poss*(a,s) and *Holds*(f,s) denote that in situation s, action a is possible and property f is known to hold, respectively. An instance of $\mathrm{Ab}(f,s)$ indicates the presence of abnormal condition f in situation s. Hence, the precondition axiom says that normally a delivery is possible if the robot carries the object in question and happens to be in the office of the recipient. However, the action fails under the unusual circumstances that the respective person is not traceable or the object has been lost.

Effects of high-level actions are specified by state update axioms, which provide a solution to the frame problem. For example, the action *Receive*(o,p) of receiving object o from person p is specified by:

$$\begin{aligned}&\textit{Poss}(\textit{Receive}(o,p),s) \supset\\ &\quad \exists p' \textit{Holds}(\textit{Request}(p,o,p'),s) \supset\\ &\qquad \textit{State}(Do(\textit{Receive}(o,p),s)) =\\ &\qquad\quad \textit{State}(s) - \textit{Request}(p,o,p') + \textit{Carries}(o,p')\end{aligned}$$

Here, the standard functions *State*(s) and *Do*(a,s) denote, respectively, the state in situation s and the situation reached after performing action a in situation s. Hence, the axiom describes the subsequent state in terms of an update of the current state by the negative effect *Request*(p,o,p') and the positive effect *Carries*(o,p'). That is, upon receiving o from p addressed to person p', the robot carries the object and the corresponding delivery request is cancelled. The effects of sensing actions are specified by so-called knowledge update axioms, an example of which will be given in Section 4.2.

Actions sometimes fail to produce the intended effect. For example, in exceptional cases a delivery may leave the recipient with the wrong item:

$$\begin{aligned}&\textit{Poss}(\textit{Deliver}(o,p),s) \supset\\ &\quad \neg\exists o' \mathrm{Ab}(\textit{Deliver}(o',p),s) \wedge\\ &\qquad \textit{State}(\textit{Do}(\textit{Deliver}(o,p),s)) =\\ &\qquad\qquad \textit{State}(s) - \textit{Carries}(o,p)\\ &\quad \vee\\ &\quad \exists o',p' \mathrm{Ab}(\textit{Deliver}(o',p),s) \wedge\\ &\qquad \textit{Holds}(\textit{Carries}(o',p'),s) \wedge o \neq o' \wedge p \neq p' \wedge\\ &\qquad \textit{State}(\textit{Do}(\textit{Deliver}(o,p),s)) = \textit{State}(s)\\ &\qquad\qquad -\textit{Carries}(o',p') + \textit{Request}(p,o',p')\end{aligned} \tag{2}$$

The condition $\mathrm{Ab}(\textit{Deliver}(o',p),s)$ represents the abnormal case of delivering the wrong item o' to person p in situation s. In this case, initiating *Request*(p,o',p') will remedy the confusion. Abnormal conditions in state update axioms, too, are assumed away by default but may serve as explanation for observed discrepancies between the expected and the actual outcome.

Possible indirect effects of actions are specified by causal relationships, which solves the ramification problem. For example, suppose the robot searches for an object o among the group of people in some room r. Whenever *Has*(p',o) becomes true, stating that person p' is in possession of the object, then all other previously considered possibilities of people having o are ruled out:

$$\begin{aligned}&\textit{Holds}(\textit{MightHave}(p,o,r),s) \supset\\ &\qquad \textit{Causes}(\textit{Has}(p',o), \neg\textit{MightHave}(p,o,r),s)\end{aligned}$$

Here, the standard macro definition *Causes*(e,r,s) means that effect e causes indirect effect r in situation s.

Explaining Action Failures

Action failure is explained on the basis of the various abnormalities that have been specified for each action. Following the solution to the qualification problem developed in [Thielscher, 2001b], abnormal conditions $\mathrm{Ab}(f,s)$ are assumed away by default unless there is evidence to the contrary. We use non-monotonic default theory to this end. Whenever the observations suggest a discrepancy between the default expectations and the actual world, the default theory entails that one or more default assumptions no longer hold.

Suppose, for example, a delivery action cannot be performed although the robot carries the right object and is in the right office. Precondition axiom (1) then offers two explanations by means of the respective positive instances of Ab. In addition, if the failed action occurs after other deliveries, then update axiom (2) offers a further explanation, namely, that the object was previously accidentally delivered to the wrong person. If so, the update axiom implies that the regular precondition *Holds*(*Carries*$(o,p),s)$ in (1) does actually not hold. In this way, the high-level controller uses its reasoning facilities to generate suitable explanations for the encountered failure [Martin and Thielscher, 2001]. By appealing to prioritised default logic [Brewka, 1994], one can specify qualitative knowledge of the relative likelihood of the various explanations for abnormal qualifications. The accompanying concept of preferred extensions then helps selecting the most likely explanation. In cases where it is impossible to provide an exhaustive specification of the reasons for a particular action to fail, a special Ab instance can be added to the precondition axiom indicating an inexplicable failure. If this abnormality is specified as being least preferred, then the controller falls back upon it only if all other explanations fail.

Planning

A controlling mechanism that monitors action execution inevitably requires a high amount of reactivity. On the low level, a set of interacting behaviours seem to meet this requirement. In an analogous manner, an abstract task planner should be able to react on events that might affect the current agenda. Action failures and general world changes require replanning for both successful accomplishment and efficiency reasons.

The maintenance of a state S and a set of abstract state evaluation functions $E : \mathcal{S} \rightarrow \mathcal{A}$ that are consulted during every action-execution cycle, are the base for the planning loop. Once a critical world change is realised, the current agenda is dropped and the planner is invoked again. The planner investigates the current state according to various criteria of decreasing priority:

```
loop(S,Z):-
  ( /* Stationary */
    notify_reach(Z, URL)    -> As = URL;
    call_help(P,Z)          -> As = [call_help(P)];
    search_fail(O,P,Z)      -> As = [email(O,P)];
    delivery(O, P, Z)       -> As = [deliver(O,P)];
    receipt(O, P, Z)        -> As = [receive(O,P)];
    /* Knowledge Acquisition */
    search(SearchDialog,Z)  -> As = SearchDialog;
    /* Navigation */
    continue(GoAct, Z)      -> As = [walk_on|GoAct];
    /* Otherwise */            As = [idle]
   ),
   execute(As, S, Z).
```

The predicates on the left hand side of the arrows can be understood as diagnostic functions of the current state. Predicate *NotifyReach* holds if people became out of reach recently. The second argument *URL* is an action sequence of notifications about the unreachable persons. Predicate *CallHelp* succeeds if it is necessary in the current state to call for help. *SearchFail* launches an email notification to the originator of a search request if none of the possible candidates has the desired item. If delivery or receipt is possible, the according actions are performed. If no stationary action is launched, that is, if all the previous state diagnostic predicates failed, then predicate *Continue* is invoked. Depending on the current state (e.g. the position), a rather complex path planning procedure is invoked and returns, if possible, a navigation action sequence (consisting of *Goto*, or several *Enter* and *GotoDoor* actions). If all else fails, the robot goes *Idle*. If stationary actions are possible, then these are executed. The plan can either be a single action or an action sequence.

Predicate *Continue* computes an optimal path plan given the current tasks and situation. In the examples below, we illustrate the use of temporal information by means of knowledge about changing doors. Keeping track of multiple requests, our delivery robot has to serve a number of people in general. For finding an optimal plan, one usually has to consider a number of aspects. While there are general aspects like high efficiency and low risk, domain dependent criteria like guarantee of service also constrain planning. Regarding navigation, we impose a ranking among the destination locations.

The planning strategy behind predicate *Continue* is amended to exploit temporal knowledge about changing door states and blocked doors in the following way: First, for all possible routes to a destination, the planner prefers a path that involves doors currently known to be open and reachable. Alternatively, a valid route is found if it does not contain doors that are known to be closed or blocked as the second choice. In the remaining case, no door is excluded from the route plan. Of course, a sensing action executed in front of a door will provide the knowledge of the state of this door at execution time just before entering the corresponding room. In other words, we prefer routes the robot recently found to work out. Please note, that no option is ruled out, but rather the best one is scheduled first.

4 Extensions for Modelling a Dynamic World

To consider also the dynamic properties of the world in action planning and reasoning, this information must be part of the robot's state description. Consequently, we have to distinguish between recent and outdated information, because otherwise invalid knowledge may mislead the robot or prevent it from finding a plan at all since some predicates form part of the action preconditions. In the following, we show how temporal knowledge about dynamic properties of the world can be represented in FLUX, together with a mechanism for its maintenance.

4.1 Representing Temporal Information

In general, information about a certain property, P, of the world is represented in the FLUX state description, Z, in three ways: If P or $\neg P$ is present in Z, then this is known to hold; otherwise nothing is known about the truth of P in Z. We represent temporal knowledge about a dynamic property by attaching the time of its observation to the corresponding fluent in Z, such that temporal as well as time-independent fluents comprise the world model.

Decay of information is simulated by forgetting about outdated fluents after a certain amount of time. By retracting such fluents that are considered to be out of date regularly, we make sure that the current state description only contains recent knowledge. Hence, no additional checks for recency of fluents are required for reasoning. Nevertheless, any observation made in the past is crucial for explaining action failures as they directly influence the planner's computation, and must be kept in the situation history.

In general, the sensory processing modules operate in two modes: autonomous broadcasting of relevant information (concurrently to the execution of actions) and a direct query of the current belief – both will be described next.

4.2 Gathering Information while Executing a Plan

Suppose a dynamic world in which doors might be open or closed and unknown obstacles might block the way. A possible knowledge update axiom (KUA) for such a domain is:

$$
\begin{array}{l}
\mathit{Poss}(\mathit{SenseDoor}(d), s) \supset \\
\quad \mathrm{Ab}(\mathit{SenseDoor}(d), s) \supset \\
\quad\quad \mathit{KState}(\mathit{Do}(\mathit{SenseDoor}(d), s), z') \equiv \mathit{KState}(s, z') \vee \\
\quad \neg\mathrm{Ab}(\mathit{SenseDoor}(d), s) \supset \\
\quad\quad \exists z \Big(\mathit{KState}(s, z) \wedge (\exists z'', t_1\, \forall t_2, t_3, t_4)\, \big(\mathit{Time}(t_1) \wedge \\
\quad\quad\quad z'' = z - \mathit{Op}(d, t_2) \circ \mathit{Cl}(d, t_3) \circ \mathrm{Ab}(\mathit{Rea}(d, t_4)) \wedge \\
\quad\quad\quad \big[\mathit{Holds}(\mathit{Op}(d, t_2), s) \wedge z' {=} z'' {+} \mathit{Op}(d, t_2) \vee \\
\quad\quad\quad\quad \mathit{Holds}(\mathit{Cl}(d, t_3), s) \wedge z' {=} z'' {+} \mathit{Cl}(d, t_3)\big]\,\big)\Big)
\end{array}
$$

where *Op*, *Cl*, *Rea* abbreviate *DoorOpen*, *DoorClosed* and *Reachable*, respectively. $\mathit{KState}(s, z)$ denotes that the robot considers z a possible state in situation s. Predicate $\mathit{Time}(t)$ simply returns the current time t. This KUA defines *SenseDoor* to yield a nondeterministic effect with respect to amendments of the current state, but positively discards previous knowledge about the door at hand.

Once a sensory processing module gained sufficient confidence about a certain property of the environment by integrating observations over time, the new belief is announced to all listening modules including the FLUX controller. FLUX in turn interrupts the current action and incorporates the new observations in the manner of the KUA above. Since new observations may render the current plan obsolete, a new plan is computed to accommodate to the new situation. Hence, the robot always follows an optimal plan according to its goals and current observations.

4.3 Active Sensing

Besides the concurrent, autonomous broadcast of detection results, the sensory processing modules provide information on demand. The FLUX controller may request the execution of a sensing action. Subsequently, the corresponding sensing module can be queried about its current belief.

Revisiting the example above, the action *SenseDoor* asks the door sensing module about the current state of the door at hand. The result is then incorporated into the FLUX state description by means of the KUA above for changing doors. Before the robot can pass through a door, it has to obtain the current state of this door for safety reasons. While $\mathit{DoorOpen}(d, t)$ is part of the preconditions for the enterdoor action, the KUA exhibits a nondeterministic definition with unknown result at the time of planning, but is resolved by *SenseDoor* at the time of execution. Only if the queried property turned out to hold, the preconditions of subsequent actions are fulfilled and the robot can continue; otherwise FLUX tries to find an alternative plan based on the new situation.

5 Example Scenarios

The proposed system for intelligent execution monitoring has been used on a Pioneer 2 mobile robot performing delivery tasks in a cluttered office environment. The robot is given a geometrical map of the operating environment. In fact, this

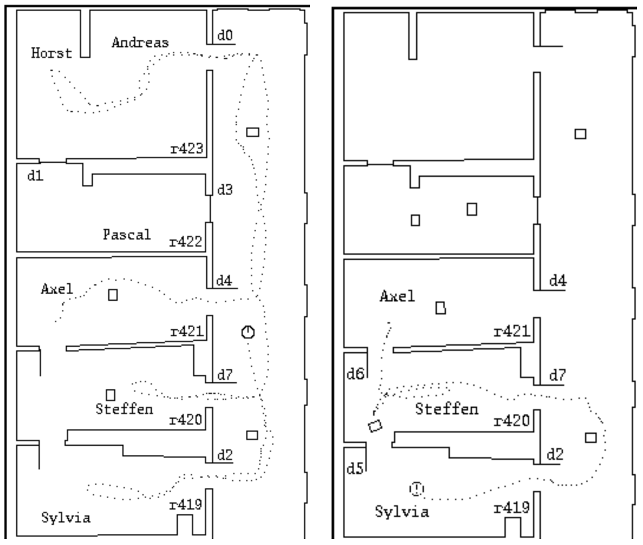


Figure 2: Office environment used in the experiments consisting of five rooms and a hallway.

assumption is imposed by the position tracking system used for navigation. Given the geometrical map, the map module extracts topological information that are subsequently incorporated into the world description in FLUX.

The aim of the experiments is to evaluate the given system with respect to its ability to deal with dynamic changes of the world and the use of intelligent execution monitoring for complex delivery tasks. As the experiments are to be performed on the real robot, the possible test scenarios are restricted by the perceptual capabilities of the robot. Unfortunately, no system for on-line tracking of persons was available to us at the time of the experiments. Therefore, we modelled the dynamic properties of the world using dynamic obstacles blocking the way of the robot, as it may be caused by people or moved pieces of furniture, using doors that open and close during the robot's operation, and using recipients that change their location unexpectedly.

In the following, we present execution traces from two selected delivery tasks. Solely for the purpose of visualisation, the examples were recorded in a simulator. However, the system's functionality has been tested in very similar situations on the real robot as well. With respect to the scenarios presented below, we noticed no fundamental differences between the performances in simulation and on the real robot.

5.1 Dealing with Temporal Knowledge

In the following, we exemplify our statement that maintaining knowledge about the dynamic properties of the operating environment is necessary in practise. The example also demonstrates how an agent can exploit the notion of recency of information.

Consider the situation depicted in the left-hand side of Figure 2. In this example, the robot is initially situated in room `r412`, at Axel's place, and it knows that door `d2` is closed. Upon receiving a request to bring a cup of coffee from Horst to Sylvia, the high-level planner is invoked. The lack of (recent) knowledge on door states prevents finding a valid path considering only doors that are known to be open. Following the second heuristic, the planner computes a valid plan excluding doors that are known to be closed:

```
State: [request(horst,coffee,sylvia),door_open(d2,24),
 in_room(r421),at(axel)|Zp]
Agenda: [walk_on,gotodoor(d4),sense_door(d4),enter(corr),
 gotodoor(d0),sense_door(d0),enter(r423),r_goto(horst)]
```

Hereof, the second argument of predicate `door_open` denotes the time of the observation in seconds. While the robot travels, the low-level behaviours bypass obstacles on its way autonomously. The interaction within the system architecture is described in more detail in the second example.

Active sensing. Having arrived at door `d4`, a sensing action explicitly determines the state of this door by querying the door sensing module. If it gained enough confidence on the door state, this observation is returned to the cognitive level, which in turn updates its state description, Z. Since there was no previous knowledge about the state of the door at hand, Z is enriched by this observation with a time stamp attached. Now knowing that the door ahead is open, the preconditions for action `enter(corr)` are satisfied and the robot continues executing the current plan.

Concurrent sensing. At door `d3`, the door sensing module autonomously signals that `d3` is open upon which FLUX computes a new plan since any observation may influence the current plan. Here, the new plan equals the continuation of the previous one due to independence of the state of door `d3`, yielding:

```
State: [door_open(d3,106),door_open(d4,79),in_room(corr),
 request(horst,coffee,sylvia),door_open(d2,24)|Zp]
Agenda: [walk_on,gotodoor(d0),sense_door(d0),enter(r423),
 r_goto(horst)]
```

Planning with multiple requests. While on its way to `r423`, the robot receives a request by Steffen, where he requires a book which should be on Andreas' desk. Since both possible providers, Horst and Andreas, can be reached (by default) according to the agent's knowledge, the planner schedules Andreas, being the nearest person, as its first destination while keeping track of remaining requests. Next, sensing the state of door `d0` provides the prerequisites to enter room `r423`. Having arrived at Andreas' desk and subsequently received the book for Steffen, the robot derives a new plan and determines to pick up the `coffee` from Horst next. On the way to Horst, the door sensing module signals that door `d1` appears to be closed. While the new observation does not interfere with the current plan, the robot could derive that it cannot enter room `r422` given its current knowledge on door states. After the robot successfully received the `coffee` from Horst, two deliveries are due, from which the planner selects Steffen's desk according to the strategy of serving the nearest person that is reachable. While passing by door `d1`, the door sensing module detects `d1` to still be closed. By means of the knowledge update axiom for `sense_door` (cf. Section 4.2), previous knowledge about `d1` is discarded and the new observation is asserted. On its way to Steffen, the robot observes that `d3` is now closed.

Exploiting temporal knowledge. As Sylvia is still waiting for her `coffee`, the robot successfully delivered the book to Steffen. Then, the FLUX planner computes a path to Sylvia and succeeds considering only such doors that are known to be open due to recently entering through `d7`, while `d2` was known to be open in the initial situation. Although directly entering room `r419` would be to the shortest path, the planner prefers paths through doors that were open recently. In the current implementation, the lifetime of temporal information is an interval of a fixed size which is sufficient for the robot to travel to the remote end of the hallway and back.

After our diligent robot delivered the `coffee` to Sylvia, Pascal requests the stapler from Axel. The planner selects a path through the corridor instead of risking the encounter of a closed door on the shortcut to Axel via Steffen's office. The corresponding figure illustrates the current situation. Meanwhile FLUX forgets about the open door `d0` and also about the closed doors `d1` and `d3`. This immediately enables the robot to start the delivery of the journal to Pascal as the state description shows:

```
State: [carries(axel,journal,pascal),at(axel),
 door_open(d4,699),in_room(r421),door_open(d2,637),
 door_open(d7,468)|Zp]
Agenda: [walk_on,gotodoor(d4),sense_door(d4),enter(corr),
 gotodoor(d3),sense_door(d3),enter(r422),r_goto(pascal)]
```

Without a mechanism for forgetting, outdated knowledge may result in invalid world descriptions that can mislead planning or even prevent the agent to find a plan. This is the case because fluents representing dynamic properties contribute to action preconditions. In the given solution for changing doors, the reachability of rooms is checked again after some time.

5.2 Explaining Unexpected Situations

In the following, we want to exemplify the application of the reasoning capabilities of FLUX together with the layered scheme of execution monitoring to finding possible explanations for action failure and an appropriate recovery strategy thereafter.

Starting at Axel's place and knowing nothing about dynamic properties of the world, the robot receives requests from Axel to bring `book1` to Steffen and `book2` to Sylvia. The right-hand part of Figure 2 depicts the example. The FLUX planner chooses to visit Steffen first according to the built-in strategy, taking the shortcut route through door `d6`. Steffen being delighted about the book, the robot plans to proceed to Sylvia carrying the remaining book by taking the shortcut through door `d5`.

Computing an explanation. Since an unexpected obstacle blocks the way to door `d5`, the *GoToPos* behaviour fails to reach the desired door node. Upon this crucial incident, the reactive controller informs FLUX about the failure together with lists of nodes that have been reached so far and remaining nodes of the topological path. By that, FLUX determines the high-level action that failed and introduces an uninstantiated abnormality fluent. Unification of the abnormality with the appropriate state update axiom for this action yields the specific abnormality that has occurred, `ab(reachable(d5,r420),51)` telling that `d5` could not be reached from within room `r420` at time 51 seconds. Given the explanation for action failure, re-planning determines a recovery strategy automatically by chosing the most preferred action from among the applicable ones – in particular, it computes an alternative path plan via the corridor.

Both doors d7 and d2 turn out to be open so that our busy robot can reach Sylvia. This situation is depicted in the right-hand part of Figure 2 and comprises:

```
State: [at(sylvia),door_open(d2,208),in_room(r419),
 door_open(d7,152),door_open(d6,112),
 carries(axel,book2,sylvia),ab(reachable(d5,r420),51)|Zp]
Agenda: [deliver(book2,sylvia)]
```

Please note that abnormality fluents are considered as dynamic properties of the world that may change, which is why a time stamp is attached in order to take exogenous world changes into account.

Reasoning about action failure. Subsequently, the robot arrives at Sylvia's desk. Surprisingly, she rejects the delivery of the book. Searching for an explanation for the failure of deliver(book2,sylvia), an ordered preference list of abnormalities is consulted. In our example, the domain axiomatisation contains the abnormalities *Deliver*, *NotLost*, *Traceable*, and *Mobility*. The applicability of the individual abnormalities is checked by analysing the history of previous action executions. This search is performed backwards in time as we consider short-term consequences of actions to be more likely than long-term consequences. The search is stopped as soon as a suitable combination is found. Axiom (2) yields the explanation that a person could have accidentially mixed up similar items. Steffen having taken book2 instead of book1 in the given example serves as a possible explanation here as it denies the executability of action deliver(book2,sylvia). In accordance to the explanation found and the sub-goal at hand, a recovery strategy is selected. In particular, the robot decides to deliver book1 – that is now supposed to be in the tray – to Steffen and to pick up book2 for Sylvia there. The corresponding state description is:

```
State: [request(steffen,book2,sylvia),at(sylvia),
 door_open(d2,208),in_room(r419),door_open(d7,152),
 ab(reachable(d5,r420),51),door_open(d6,112),
 carries(axel,book1,steffen)|Zp]
Agenda: [walk_on,gotodoor(d2),sense_door(d2),enter(corr),
 gotodoor(d7),sense_door(d7),enter(r420),r_goto(steffen)]
```

Avoiding the blocked path and using the doors that were recently open, the robot carries book1 to Steffen which he regrets to have mixed up. By that, the explanation proves to be correct. Otherwise, book2 being lost would serve as an alternative explanation. Eventually, book2 is successfully delivered to Sylvia.

It is important to note that we demonstrated both aspects, dealing with temporal information and explaining action failures, in separate examples for the purpose of understanding only, while they are employed simultaneously in practise.

5.3 Applicability

Given a suitable set of low-level behaviours and sensory processing modules, the cognitive controller needs to be adapted to the tasks at hand. Please note, for the sake of general applicability, the domain axiomatisation solely resides within FLUX, the cognitive controller. Therefore, the solution can be easily extended or adapted for different scenarios.

6 Conclusions and Future Work

We have presented a layered scheme of execution monitoring for mobile robots operating in known structured environments, allowing for action failures due to the interaction with humans and dynamic changes in the environment. It allows the robot to find explanations for action failure using a logic-based world model and the history of the task execution. In the experiments, we were able to illustrate the advantages of reasoning-based execution monitoring over reactive control methods. The implementation is based on the fluent calculus along with its augmentations.

Our approach is similar to the systems by [Shanahan, 1996; Haigh and Veloso, 1997; Hähnel *et al.*, 1998] in the sense that all use high-level planning on real robots. Shanahan's Khepera robot based on the event calculus makes heavy use of abductive planning and explanations of failures. However, these failures are mainly due to the sensor limitations of this fairly simple robot. In contrast to our work, the high-level planner is not embedded in a complex modular architecture with clearly defined low-level behaviours such as obstacle avoidance and sophisticated localisation techniques.

The control architecture used in our work is more comparable to the ones proposed in [Haigh and Veloso, 1997] and [Hähnel *et al.*, 1998]. These, in turn, provide only hand-coded recovery procedures for failures of the current action. Undetected failures of earlier actions are not considered. Furthermore, the use of the fluent calculus allows us to model side effects of actions.

Preference lists for action failures involve a great deal of speculation and need to be specified in advance. An alternative method would be a form of hypothesis testing: being left with a set of possible explanations for some action failure, each of them could be double checked by additional sensing/state verification. This topic is closely related to central questions in the field of active perception, for example, active vision. The application of a cognitive planner for sensing actions in the fashion outlined above could help in minimising action effort and maximising knowledge gain. Thus, it seems promising to formalise active vision domains within the fluent calculus. Furthermore, the static nature of the preference list could be overcome by learning from experiences.

References

[Brewka, 1994] G. Brewka. Adding priorities and specificity to default logic. In *Proc. of the European Workshop on Logics in AI (JELIA-94)*, volume 838 of *LNAI*, pages 247–260. Springer, 1994.

[Fernández and Simmons, 1998] J. L. Fernández and R. G. Simmons. Robust execution monitoring for navigation plans. In *Proc. of the Conf. on Intelligent Robots and Systems (IROS-98)*, 1998.

[Fox *et al.*, 1999] D. Fox, W. Burgard, and S. Thrun. Markov localization for mobile robots in dynamic environments. *Artificial Intelligence Research*, 11:391–427, 1999.

[Gat, 1992] E. Gat. Integrating planning and reacting in a heterogenous asynchronous architecture for controlling real-world mobile robots. In *Proc. of the 10th National*

Conf. on Artificial Intelligence (AAAI-92), pages 809–815, 1992.

[Giacomo and Levesque, 2000] G. De Giacomo and H. Levesque. ConGolog, a concurrent programming language based on the situation calculus. *Artificial Intelligence*, 121(1–2):109–169, 2000.

[Giacomo *et al.*, 1998] G. De Giacomo, R. Reiter, and M. Soutchanski. Execution monitoring of high-level robot programs. In *Proc. of the Int. Conf. on Principles of Knowledge Representation and Reasoning (KR-98)*, pages 453–465, 1998.

[Hähnel *et al.*, 1998] D. Hähnel, W. Burgard, and G. Lakemeyer. GOLEX - Bridging the gap between logic (GOLOG) and a real robot. In *Proc. of the 22nd German Conf. on Artificial Intelligence (KI-98)*, 1998.

[Haigh and Veloso, 1997] K. Z. Haigh and M. M. Veloso. High-level planning and low-level execution: Towards a complete robotic agent. In *Proc. of First Int. Conf. on Autonomous Agents*, pages 363–370, 1997.

[Martin and Thielscher, 2001] Y. Martin and M. Thielscher. Addressing the qualification problem in FLUX. In *Proc. of the German Annual Conf. on Artificial Intelligence (KI-2001)*, pages 290–304, 2001.

[Martin, 2003] Y. Martin. The concurrent, continuous FLUX. In *Proc. of the 18th Int. Joint Conf. on Artificial Intelligence (IJCAI-2003)*. Morgan Kaufmann, 2003.

[Reiter, 1980] R. Reiter. A logic for default reasoning. *Artificial Intelligence*, 13(1-2):81–132, 1980.

[Reiter, 1996] R. Reiter. Natural actions, concurrency and continuous time in the situation calculus. In *Proc. of the Int. Conf. on Principles of Knowledge Representation and Reasoning (KR-96)*, pages 2–13, 1996.

[Saffiotti, 1993] A. Saffiotti. Some notes on the integration of planning and reactivity in autonomous mobile robots. In *Proc. of the AAAI Spring Symposium on Foundations of Planning*, pages 122–126, Stanford, CA, US, 1993.

[Sandewall, 1989] E. Sandewall. Combining logic and differential equations for describing real-world systems. In *Proc. of the Int. Conf. on Principles of Knowledge Representation and Reasoning (KR-89)*, pages 412–420, 1989.

[Shanahan, 1990] M. Shanahan. Representing continuous change in the event calculus. In *Proc. of the European Conf. on Artificial Intelligence (ECAI-90)*, pages 598–603, 1990.

[Shanahan, 1996] M. Shanahan. Robotics and the common sense informatics situation. In *Planning with Incomplete Information for Robot Problems: Papers from the 1996 AAAI Spring Symposium*, pages 95–106, 1996.

[Stentz, 1995] A. Stentz. The focussed D* algorithm for real-time replanning. In *Proc. of the 14th Int. Joint Conf. on Artificial Intelligence (IJCAI-95)*, pages 1652–1659, 1995.

[Thielscher, 2001a] M. Thielscher. The concurrent, continuous fluent calculus. *Studia Logica*, 67(3):315–331, 2001.

[Thielscher, 2001b] M. Thielscher. The qualification problem: A solution to the problem of anomalous models. *Artificial Intelligence*, 131(1-2):1–37, 2001.

[Thielscher, 2002] M. Thielscher. Programming of reasoning and planning agents with FLUX. In *Proc. of the Int. Conf. on Principles of Knowledge Representation and Reasoning (KR-2002)*, pages 435–446, 2002.

[Thrun *et al.*, 2000] S. Thrun, D. Fox, W. Burgard, and F. Dellart. Robust Monte Carlo localization for mobile robots. *Artificial Intelligence*, 128(1-2):99–141, 2000.

An Extended Panorama: Efficient Qualitative Spatial Knowledge Representation for Highly Dynamic Environments

Thomas Wagner
Center for
Computing Technologies
University of Bremen
twagner@tzi.de

Christoph Schlieder
University of Bamberg
christoph.schlieder@wiai.
uni-bamberg.de

Ubbo Visser
Center for
Computing Technologies
University of Bremen
visser@tzi.de

Abstract

Qualitative spatial knowledge representation has gained much attention in recent research and has resulted in several successful applications focused on static environments. In this paper, we argue that dynamic environments with physical robots impose different requirements on spatial representations. We introduce a novel approach based on ordering information and metric representations: an *extended panorama*. We show how this representation can be used in order to describe behavior in a flexible dynamic manner and how updating and maintaining a world model can be done efficiently. We demonstrate the performance of the approach in the RoboCup domain and discuss further issues.

1 Introduction

One of the major difficulties for planning and coordinating (realtime) robotic behavior in physical, dynamic environments is spatial reasoning. We claim that qualitative spatial reasoning can be used to reduce many of these problems.

Qualitative spatial knowledge representation has gained strong interest in the last decade and has been used successfully in a wide range of applications (i.e., traffic monitoring, geographical-information systems [Cohn *et al.*, 2000], [Cohn and Hazarika, 2001]). To date, however, approaches to qualitative knowledge representation, when used in robotics, often have focused on qualitative representation of more or less static environments (e.g., rooms, buildings). It has been shown that these environments can be adequately described in terms of qualitative spatial knowledge i.e., topological descriptions and metrical information [Kuipers, 2000], [Yeap and Jefferies, 1999]. For both types of qualitative knowledge, as well as for ordinal knowledge, expressive representations with powerful inference mechanisms have been developed. But the requirements for dynamic and realtime environments however differ significantly from those in static environments. We propose therefore a qualitative spatial representation, an *extended panorama*, based on ordering information optionally extended with metric descriptions. We argue that an *extended panorama* is expressive enough to represent a large set of relevant spatial descriptions which are required to describe complex, coordinated behavior in *dynamic* and real-time environments, and motivate that an *extended panorama* can be maintained and updated efficiently.

2 Motivation

Several difficulties in planning and coordinating robotic behavior in dynamic, realtime environments are concerned with spatial representations and reasoning. While qualitative spatial knowledge in static environments provides a stable basis and can be used in different courses of actions at different points of time, the use of spatial knowledge in dynamic environments is often restricted to temporary, immediate use. As a result, qualitative spatial knowledge representation in dynamic environments have a different focus.

In the following we describe some of the most relevant problems and the resulting requirements for qualitative spatial knowledge representations/-reasoning.

Flexible spatial representations

Modeling individual behavior of a single agent, especially for highly dynamic situations, is very often done from an egocentric point of view. Since the world model is based on individual perception, an egocentric representation of the world can be maintained and updated more easily than a global, allo-centric view. This is covered by many texts of didactic literature on learning tactical and strategic behavior [Lucchesi, 2001]. In driving lessons, for instance, spatial descriptions are very often described in terms like *left, right, front, behind*. The same evidence can be found in articles on sport training like soccer. Furthermore, these descriptions are not only ego-centric but often they are also quite abstract and do not depend on a specific orientation or even a defined position. This means that spatial descriptions, especially from an egocentric point of view, are often invariant with regard to translation and rotation. A big advantage is that these levels offer a great flexibility in reusing behavior patterns in very different situations. This abstraction is also reasonable for robotic agents on heterogeneous platforms and for robots operating in different domains as it allows the reuse of already described and established behavior patterns.

Considering multiagent-behavior, an at least *pseudo*-allocentric representation is required in order to coordinate tactical and strategic group behavior. Furthermore, individual ego-centric views have to be mapped efficiently into an allocentric view.

Another important requirement is imposed by the fact that perceived spatial data is subject to different sources of error and imprecision depending on factors like vision processes, lightning conditions, and sensory hardware. Thus qualitative spatial knowledge representation should support representations on different levels of granularity, i.e., depending on the source of information (and therefore the quality,) a fine grained measurement should be chosen for precise and certain information and a coarse one for less precise data[1].

Interoperability and reusability of behavior models

The development of behavior models is a complex, time-consuming task and therefore reusability is an important requirement. A typical example for complex behavior patterns is the RoboCup domain. Modeling behavior based on real world soccer tactics and strategies is a tedious and timeconsuming task and requires in-depth expert knowledge. Although most of the basic tactical behaviors are not only similar within a specific league but also between different leagues the behavior usually has to be re-modelled for each team. This strongly relies on quantitative information, starting with primitive skills moving to complex strategic behavior. The same applies when changing the application domain and the sensory hardware. Each change often demands alternations on each level of the behavior model.

Applying spatial least commitment strategies

Planning cooperative behavior is usually directly coupled with the application of least commitment strategies (least commitment means that decisions during the planning process should be made as late as possible, usually by incrementally evaluating time-/ordering constraints.). This is done in at least two aspects:

- least commitment during planning process and
- in the representation of the result of the planning process.

In both scenarios, time constraints are resolved as late as possible. This strategy helps to avoid early conflicts during the planning process and a more flexible use of the calculated plan due to partial ordering (in comparison to total-order plans). Especially in dynamic multi-agent-systems (MAS) (based on execution and monitoring) least commitment strategies play a crucial role. Particularly in physical domains least commitment strategies should also be applied on spatial constraints. In soccer-domain, for example, we should be able to decide as late as possible whether we intend to play a left or right wing attack. This requires spatial representations to be invariant, e.g., according to translation and rotation depending on the intended behavior.

The remainder of the paper is organized as follows: in section 3 we describe the concept of a panorama introduced by Schlieder [Schlieder, 1996]. In the following section 4 we show different ways how to extend the panorama and how they can be used at different levels of granularity in different environments (e.g., on robots with different perceptions). In section 5 we show how a *extended panorama* can be used as a qualitative monitor. Finally in section 6 we show how the panorama can be applied in the RoboCup domain and give a short discussion of our first experiences and the future steps.

[1]Furthermore, a mapping between fine-grained and coarse-grained measurement is required.

3 Panorama

A lot of approaches to spatial reasoning focus on the representation of large scale space. Large scale space can for instance be defined as *„space (that) is a significantly larger scale than the observations available at an instant"* [Kuipers and Levitt, 1988]. Based on these complex representations different powerful inference methods have been developed which allow reasoning on cardinal directions [Frank, 1996], distance representations (e.g., [Clementini *et al.*, 1997]) and topological representations (e.g., [Cohn *et al.*, 1997]) on qualitative spatial representations. Most of these approaches, however, are based on rather static environments.

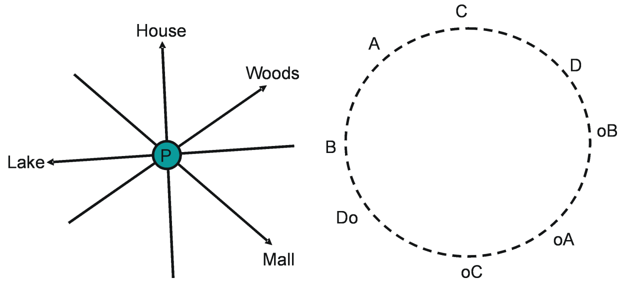
Figure 1: Highly dynamic environments: traffic, sports (e.g., RoboCup)

As a result, knowledge-based systems with spatial reasoning focus on domains like geographical information systems (GIS). When addressing domains with highly dynamic agent behavior ranging from sports (e.g., soccer, football) to traffic qualitative spatial knowledge is required for modeling complex behavior. This implies different levels of granularity in order to be able to plan complex coordinated behavior on different levels of abstractions (playing a pass vs. playing offside). Furthermore, the representation has to be updated efficiently allowing us to recognize relevant changes in the environment and adopt our model of behavior.

The panorama approach

The concept of panorama representation has been studied extensively due to to specialized sensors (e.g. ommnivision) (see e.g. [Bourque and Dudek, 1998], [Zheng and Tsuji, 1992]). In this paper we present a different approach.

A complete, circular panorama can be described as a 360^0 view from a specific, observer-dependent point of view. Let *P* in figure 2(a) denote a person, than the panorama can then be defined as the strict ordering of all seen objects: *house*, *woods*, *mall*, *lake*. This ordering, however, does not contain all ordering information as described by the scenario 2(a). The *mall* is not only directly between the *woods* and the *lake*, but more specifically between the opposite side of the *house* and the *lake* (the tails of the arrows). In order to represent the spatial knowledge described in a panorama scenario, Schlieder [Schlieder, 1996] introduced a formal model of a panorama.



(a) Concrete panorama (b) Abstract panorama

Figure 2: Panorama-views

Definition 3.1 (Panorama) *Let* $\Theta = \{\theta_1, \ldots, \theta_2\}$ *be a set of points* $\theta \in \Theta$ *and* $\Phi = \{\phi_1, \ldots, \phi_n\}$ *the arrangement of n-1 directed lines connecting* θ_i *with another point of* Θ*, then the clockwise oriented cyclical order of* Φ *is called the panorama of* θ_i*.*

As a compact shorthand notation we can describe the panorama in figure 2(b) as the string $< A, C, D, Bo, Ao, Co, Do, B >$. Standard letters (e.g., A) describe reference points and letters with a following *o* (e.g., Ao) the opposite side (the tail side). Since the panorama is a cyclic structure the complete panorama has to be described by n strings with n letters, with n the number of reference points on the panorama. In our example, the panorama has to be described by eight strings. Furthermore, the panorama can be described as a set of simple constraints $dl(vp, lm_1, lm_2)$[2]. Based on this representation, Schlieder [Schlieder, 1993] also developed an efficient qualitative navigation algorithm.

Applying the panorama representation to competitive and cooperative scenarios, we can furthermore infer groups of competitive and/or cooperative agents, given we are able to distinguish them physically, i.e the basic panorama has to be differentiated slightly to $P_{basic} =< O_{pponent}, O_{wn}, OC_{dl} >$, with $O_{pponent} \in P$ and $O_{wn} \in OP$ the set of observed points and OC_{dl} a set of *direct-left*-ordering constraints.

One of the major difficulties in planning complex coordinated behavior in dynamic, physical environments is to identify stable situations. This means we want to describe behavior on different levels of granularity and we want to identify situations that are similar enough to apply planned behavior. As a consequence, spatial representations should abstract from irrelevant details. One way to achieve this is to use a representation, like a panorama specified above, which is invariant according to rotation and translation. Applied to the RoboCup domain such a representation would allow us to describe behavior that is on the one hand independent of the (exact) location of an agent on the field and that is on the other hand invariant to the orientation of the agent. Figure 4.1shows a situation with a concrete attack of the white team. Describing behavior for this situation relies in the orientation of the configuration of observed points but does not rely on the specific position according to the length of the field, i.e., even if the same configuration is found 10 meters behind the current position, the same models of behavior can be applied. A more appropriate representation for this scenario is described in figure 3(b)[3].

But evidently, not every behavior can be described in such an abstract manner. In order to model complex, coordinated behavior direction information is usually involved. Additionally in some situations different metric information is required. In the following section we show how the panorama can be extended so that more detailed ordinal and metric information can be introduced.

4 Extended Panorama

Modeling complex behavior in physical environments requires different degrees of absolute and relativ qualitative spatial representations. While modeling individual behavior is usually based on ego-centric representations, for ordinal as well as for metric information, coordinated behavior has either to be modelled in allo-centric terms or requires a mapping between different ego-centric representations. Furthermore, abstract, relative (invariant) representations should support a mapping into more concrete representation according to the level of detail in the planning process. Furthermore, we show how a basic panorama can be extended with ordinal and metric information so that it can build the foundation for modeling intuitive domain-dependent spatial descriptions.

4.1 Towards Ordinal Panorama Representations

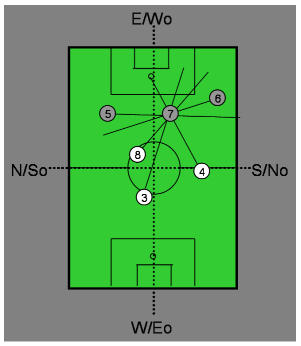
Introducing ordinal information can be done in at least two ways. Based on an ego-centric view, we can simply introduce a heading of the robot into the panorama (assuming that the robot at least knows his relative orientation according to his individual sensors). Defining the heading as *front*, we can easily introduce the *back/behind* side as the opposite direction and furthermore left and right as a 90^0 distance from the heading. Figure 3(b) shows an example leading us to the following *extended panorama* for player 7: $< 4, 5o, L, 6, 8o, 3o, B, 4o, 5, R, 6o, 8, 3, F >$. Although this representation does not provide exact ordinal information it allows to infer an ordinal interval for a given landmark θ_1 according to its ego-centric orientation $OrdEgo_4 = \{left, right, front, behind\}$[4]. In our example we know that player 5 and the opposite of player 4 are between the back- and right side. Therefore an ego-centric (pseudo-) ordinal panorama can be defined as $P_{EPO} = \{DLM, Ref_{ego}, OC\}$ (EPO) with DLM as a set of possibly dynamic landmarks, an ordinal ego-centric reference system Ref_{ego} and a set of ordering constraints OC. Since this representation integrates only ego-centric ordinal information, it is still invariant according to translation and rotation and hence is still an abstract representation which can be applied for numerous different situations.

Although an ego-centric view is often an intuitive way to model individuell behavior, an allo-centric view is necessary to describe coordinated multi-agent behavior. Multi-

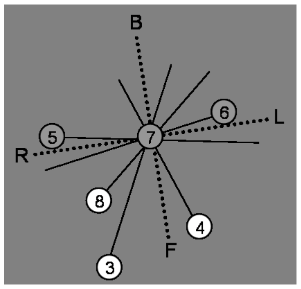
[2]Short for $direct{-}left(viewpoint, landmark_1, landmark_2)$.

[3]As a consequence, in section 4, we argue that this representation should be extended with direction information.

[4]Different ego-centric reference systems may be used at different granularities according to the requirements of the domain.



(a) Allo-centric view



(b) Ego-centric, invariant View

Figure 3: Pseudo-ordinal information

agent behavior can be described directly (e.g. shared plan approach [Grosz and Kraus, 1996] or has to be communicated in order to achieve commitment [Smith *et al.*, 1998]. As a consequence it is required that spatial knowledge can be described and communicated in an allo-centric way based on ego-centric representations. This can be achieved in two ways.

The simplest solution is to communicate landmarks according to the ordinal intervals. Based on this information, the other agent can try to map his panorama into his ego-centric panorama based on commonly observed landmarks. A problem of this approach is that it can lead to very imprecise representations since a defined landmark does not necessarily appear in the same order in a panorama from different reference points (points of view) although the observed landmarks are the same.

In order to preserve ordering constraints of a given panorama a fixed reference point is introduced. Fixed reference points have to be defined domain-dependently. In the soccer domain we may use the own or the opposite goal or the left or right intersection point of the mid line with the outer line[5]. In ego-centric views the reference-systems can be easily defined in terms of physical properties of the agent itself. Since the reference system is not based on perceptional input, it can be defined in any way as long as it preserves a 90^0 distance between the basic four directions. For an allo-centric representation the reference systems is based on perceptional input and therefore should be chosen carefully with respect to the quality and simplicity in order to perceive at least one of the reference points. Referring to the example in figure 3(a), we describe the pseudo-allo-centric, pseudo-ordinal panorama (PAPO) for player 7 as $< 4o, N, 3o, 8o, 6, E, 4, S, 8, W, 6o, 5 >$.

Although a PAPO-panorama does not provide a real allo-centric view it provides the foundation for coordinated behavior. In our example, player 7 cannot communicate his panorama to other players even though an allo-centric reference system with fixed landmarks is used since a panorama can appear completely different from different point of views.

[5]In the physical RoboCup leagues: SSL, F500 and *Sony Four Legged League* there exists a fixed set of labeled landmarks.

Nevertheless, on the basis of a PAPO-panorama, player 7 can tell e.g. another player where he wants to pass a ball or where he expects a player to move. This means that a player cannot carry out inferences which are directly based on a panorama from a different point of view.

4.2 An Ordinal Panorama with Metric Information

Based on ordering information which is extended with land marks, fixed ones for PAPO and dynamic, ego-centric ones with EPO, the panorama representation provides a wide range of qualitative information with different levels of abstraction. However, the precision of ordinal information is restricted to intervals which define the lower and upper bounds. In more concrete planning phases more precise ordinal information may be required. At least two different approaches can help to increase ordinal precision.

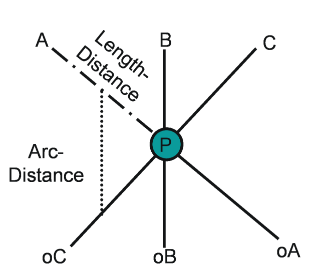
Since EPO- and PAPO-panoramas describe *ordinal intervals* (e.g. player 4 is between east and south), increasing the numbers of reference points minimizes the *ordinal intervals* and therefore increase ordinal precision. This approach is somehow limited because all fixed reference points have to be defined domain-dependently, i.e., depending on the spatial structure of the static environment of the domain it is more or less difficult to define appropriate fixed reference points. Also different requirements have to be fulfilled. Firstly, reference points should be easy to perceive from different location for all agents that participate in coordinated behavior. Secondly they should have a similar distance between each other in order to provide a uniform granularity.

Given the case that fixed reference points cannot be defined, absolute angle distance between two possibly dynamic points (i.e., agents) can be used to increase ordinal precision instead (see figure 4.2). Angle distance should be mapped on an absolute qualitative scale with equal distance. Given a fixed angle distance of 30^0 we are able to distinguish the location of an object, e.g., *north/west(NW)* and *west/north(WN)* instead of somewhere between *north* and *west*. If we are able to distinguish between 30^0 angle distance, we can even classify, e.g., between *W, WNW, NW, NNW, N*. The choice between the first and the latter approach depends on the characteristics of domains and on the quality of the perceptional input. Since both approaches result in the same panorama representation it is even possible to use both approaches simultaneously depending on the specific situation.

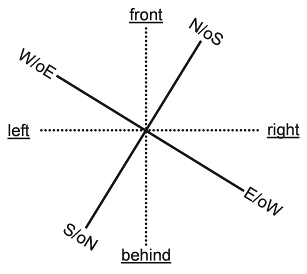
4.3 Additional extentions and variants

In order to describe complex coordinated behavior distance information between the observing agent and an observed landmark (agents or fixed landmarks) is often required. Although distance information cannot be inferred from ordering- and ordinal information, the panorama can be easily extended with distance information. Different qualitative distance representations with powerful inference mechanisms have been developed [Clementini *et al.*, 1997].

Depending on the specific domain and the specific situation different variations can be built. An *extended panorama* is not restricted to PAPO- and EPO-panorama. In different domains various combinations of PAPO- and EPO-panorama



(a) Distances in a panorama



(b) A hybrid approach

Figure 4: Different combinations of EPO- and PAPO-panoramas

can be build. While the ego-centric EPO-panorama requires a heading which is directly associated with the reference system, a PAPO-panorama does not depend on a special heading and is therefore (perhaps) more appropriate, e.g., for robots based on omnivision. Nevertheless a PAPO-panorama can easily be extended to a EPAPO-panorama with headings if a translation- and rotation independent representation is not desired (see figure 4.2) (e.g., immediate communication is required).

Another way to adapt an *extended panorama* to specific situations can be achieved by using different reference systems on different levels of granularity. For more abstract planning phases, e.g., in HTN-planning or in situations with imprecise information, more coarse information can be handled with restricted reference systems like $OrdEgo_4$ (s. above). On the other hand, in detailed planning phases or in situations where more detailed information is required a more finely grained reference system can be chosen.

5 Application of the extended panorama approach

The described panorama approach is designed to be employed in various applications. One of the requirements is that the method is fast and accurate enough to serve programs running in both real-time and dynamic environments. In the following we discuss a first application of our approach in the RoboCup domain.

In order to demonstrate the various options of our approach, a panorama test environment has been developed. Figure 5 shows what we call a ”monitor” of an actual situation in a possible soccer game. Figure 6 shows a Sony-Legged-League soccer field. Overall, the field is marked with 15 landmarks which are placed at important positions on the field (e.g., the corners, the goal area, etc.). Please note that the approach is applicable in all available leagues.

5.1 Simple situation

First, we would like to demonstrate the performance of our approach with a simple example. Figure 6 shows a part of the soccer field with two objects, the object with the angle is the soccer agent *(me)* and the other object *(Z)* is the ball. The soccer agent aims at the goal on the right hand side. If we compute this simple panorama, the method results in

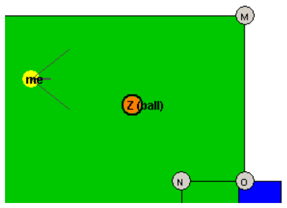


Figure 6: Simple situation with the current agent (me) and a ball

```
Z(MEDIUM,VERY_CLOSE), z(MEDIUM,VERY_FAR).
```

The first argument is the qualitative length distance. The second argument in the brackets is the qualitative angle distance between the soccer agent's straight view and the position of the object *Z*. This means that the object *Z* has a very close angle distance and is medium far away with regard to the length distance. The angle to the opposite angle distance *(z)*, however, is very far, which is obvious in this case. If we translate this onto an abstract level, we get the following:

```
The BALL is MEDIUM FRONT.
```

Please note that this is the most simple case, only considering the perception of the current agent without landmarks, requirements, and avoidance information.

If we also consider landmarks, the following panorama is computed:

```
        ( Z(MEDIUM,VERY_CLOSE),
O(FAR,VERY_FAR),N(FAR,MEDIUM), m(FAR,MEDIUM),
   z(MEDIUM,FAR),o(FAR,FAR), m(FAR,FAR),
              M(FAR,FAR) )
```

or in short

```
(Z O N m z o n M)
```

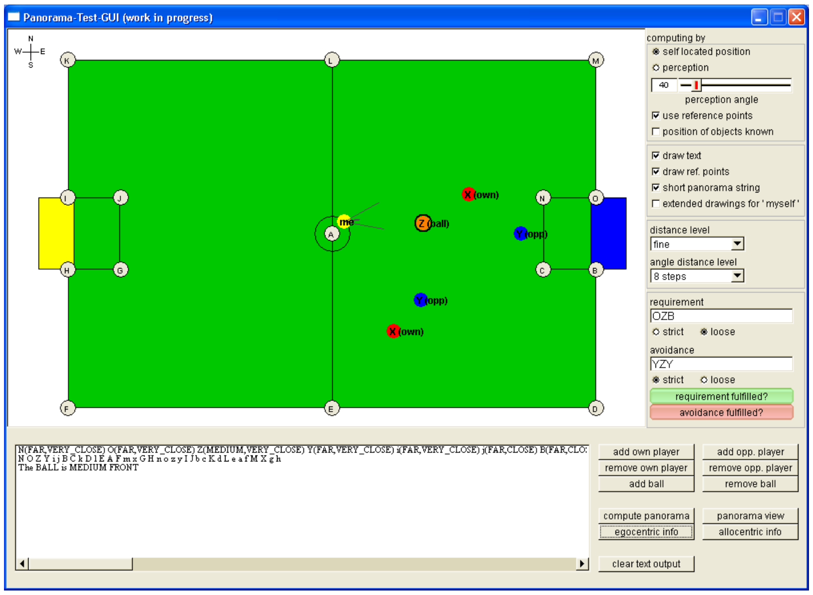
This additional information can be important in complex situations.

5.2 Complex situation

Figure 5 describes a more complex situation. We are dealing with the current player *(me)*, two own team mates denoted as *X*, two opponent players denoted as *Y*, and the ball *(Z)*. Using landmark as an additional instrument to collect and derive better information, the following panorama is computed:

```
( N O Z Y i j B C k D l E A F m x G H n o z y
       I J b c K d L e a f M X g h )
```

The computed panorama can now be used to fulfil additional queries the current agent might generate while planning the next steps. Suppose he wants to shoot at the goal and he needs the ball position between the two posts. A simple query can answer this requirement. Figure 5 shows this situation. On the right hand side the requirement *OZB* is given. This ordering information denotes that the ball *Z* should be right of the left post *landmark O* and left of the right post *landmark Z*. Since this is not a strict requirement, which means that the player might also shoot when the situation is slightly different, we give the additional information ”loose”. We can see that this requirement is fulfilled in this situation.



computing by
[✓ self located position] perception
40 perception angle
[✓ use reference points]
position of objects known
[✓ draw text]
[✓ draw ref. points]
[✓ short panorama string]
extended drawings for ' myself '
distance level: fine
angle distance level: 8 steps
requirement: OZB
strict [✓ loose]
avoidance: YZY
[✓ strict] loose

```
N(FAR,VERY_CLOSE) O(FAR,VERY_CLOSE) Z(MEDIUM,VERY_CLOSE) Y(FAR,VERY_CLOSE) i(FAR,VERY_CLOSE) j(FAR,CLOSE) B(FAR,CLO
N O Z Y i j B C k D l E A F m x G H n o z y I J b c K d L e a f M X g h
The BALL is MEDIUM FRONT
```

Figure 5: Complex situation with the current agent (me), a ball, two opponents, and two team mates

Another complex situation can be seen in figure 7. This situation consists of eight objects, i.e., the current player and three team mates, tree opponent players, and the ball.

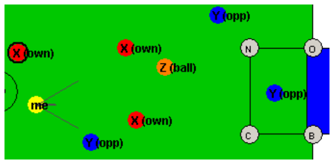


Figure 7: Complex situation with the current agent (me), three own team mates, three opponent players, and a ball

Collecting the panorama information including the landmarks results in the following:

```
( B C X y z n o y b c x Y Z N O Y )
```

Asking where the ball is and how far it is away the method provides us with

```
The BALL is MEDIUM FRONT.
```

In addition, we are able to get a *pseudo-allocentric* view. This means that the player is able to infer an allocentric view from his information. In the current situation, for instance, we can infer that the

```
The BALL is EAST of MY_SELF in MEDIUM
DISTANCE.
```

Please note that north is the top border in the figure (see also the windrose in the left corner of figure 5). Therefore, the ball is east of the player.

5.3 Additional features

The extended panorama approach has additional features which are discussed in the following. These features concern the *granularity* of the underlying reference system and the ability to *focus* on special parts or subsets of the calculated panorama. Both features directly influence the efficiency of the approach.

The granularity can be varied according to the desired outcome. It makes sense, for instance, to have a rather coarse distance level and angle distance level if the agent is planning on a higher and abstract level. An example is a counter-attack across the right wing. In the beginning, the agent might want to know whether the right wing is free of opponents or how many opponents are in that area. The area, however, can be rather big. Therefore, a coarse distance level and a 4-step angle distance level might be appropriate (please refer also to figure 4.2 for the distance explanation). A typical one-two, on the other hand, requires more precise knowledge in a smaller area. Therefore, we would choose a fine distance and a 16-step angle distance level. Our approach allows different levels of granularity in a flexible manner. The distance level can be set to fine or coarse where the distance angle level can be 4, 8, or 16 steps for the 360^0.

We are also able to focus on special parts of the panorama which has been calculated earlier. As already discussed, the PAPO-panorama gives us a 360^0 degree view. In special situations like a one-two situation we do not really need a 360^0 degree information for the next few updates (in real-time environments such as the simulation league in RoboCup the next few cycles). Instead, we are interested in *parts* of the existing panorama. Taking the panorama as calculated above: there is a lot of ordering information and we only need to focus on a subset, e.g., subset 1 in figure 8). We can now take this part and update our world model accordingly. Subset 2 in the same figure might also be worth considering, however, this could be a completely different intention. Here, the player might not want to consider a one-two situation. He might be checking out a free area to pass or to run to.

Again, this approach is flexible enough to allow these kind

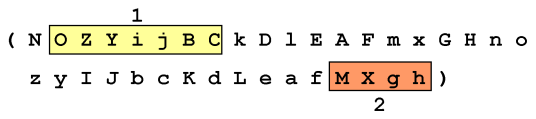

1
(N O Z Y i j B C k D l E A F m x G H n o
z y I J b c K d L e a f M X g h)
2

Figure 8: Calculated panorama, focused on two areas

of foci in order to ”save” some costly CPU time. In addition, we are able to apply the above-mentioned granularity options. This means that we can focus on different subsets of the calculated panorama with different granularities. This also supports efficiency in an impressive manner and we believe that this is an important if not necessary feature in highly dynamic and real-time environments.

6 Discussion

The use of qualitative spatial knowledge plays a crucial role in modeling and maintaining complex team behavior. First, modeling behavior knowledge in physical domains is a complex and time-consuming task and strongly relies on spatial knowledge. The *extended panorama*-approach supports abstract, rotation- und translation-invariant representations and, thus simplifies the reuse of behavior models in different domains and for robots with different hardware platforms. Second, the *extended panorama* provides a wide range of variants: An ego-centric, translation- and rotation-invariant view which can be extended in several ways in order to support ordinal information at different levels of granularity and from different sensory sources. In the same way, it supports different (pseudo-) allo-centric views. Furthermore, we motivated how the *extended panorama* provides the basis for efficient planning on several levels. The extended panorama has a compact representation, and changes can be updated simply by exchanging ordering constraints (in *extended panoramas* without metric information). Moreover, it supports least commitment strategies on spatial information which can help to minimize plan failure in dynamic planning environments. In order to validate this approach, we are currently integrating a qualitative spatial monitor based on the *extended panorama* in our agent architecture which uses a spatial least commitment strategy.

For future work we consider the following objectives. One of the main tasks is to run intensively experiments in real-time and dynamic environments. This means that we have to build different interfaces to various teams in various RoboCup soccer leagues. We will test this approach in the Simulation League, the Sony-Legged-League, and in one of the other physical robotic leagues.

Another task is the development of complex spatial predicates at different levels of granularity such as ”*a long pass in space*”. These predicates will support abstract and concrete behavior modeling. They are supposed to cover the ”language of the domain”. In addition, we have to evaluate which subset can also be used domain-independently.

References

[Bourque and Dudek, 1998] E. Bourque and G. Dudek. Automated imagebased mapping, 1998.

[Clementini *et al.*, 1997] Eliseo Clementini, Paolino Di Felice, and Daniel Hernandez. Qualitative representation of positional information. *Artificial Intelligence*, 95(2):317–356, 1997.

[Cohn and Hazarika, 2001] A G Cohn and S M Hazarika. Qualitative spatial representation and reasoning: An overview. *Fundamenta Informaticae*, 46(1-2):1–29, 2001.

[Cohn *et al.*, 1997] Anthony G. Cohn, Brandon Bennett, John Gooday, and Nicholas Mark Gotts. Qualitative spatial representation and reasoning with the region connection calculus. *GeoInformatica*, 1(3):275–316, 1997.

[Cohn *et al.*, 2000] A.G. Cohn, J. Fernyhough, and D. Hogg. Constructing qualitative event models automatically from video input. *Image and Vision Computing*, 18:81–103, 2000.

[Frank, 1996] Andrew U. Frank. Qualitative spatial reasoning: Cardinal directions as an example. *International Journal of Geographical Information Science*, 10(3):269–290, 1996.

[Grosz and Kraus, 1996] Barbara J. Grosz and Sarit Kraus. Collaborative plans for complex group action. *Artificial Intelligence*, 86(2):269–357, 1996.

[Kuipers, 2000] Benjamin Kuipers. The spatial semantic hierarchy. *Artificial Intelligence*, 119(1-2):191–233, 2000.

[Kuipers and Levitt, 1988] B. J. Kuipers and T. Levitt. Navigation and mapping in large scale space. *AI Magazine*, 9(2):25–43, 1988.

[Lucchesi, 2001] Massimo Lucchesi. *Choaching the 3-4-1-2 and 4-2-3-1*. Reedswain Publishing, edizioni nuova prhomos edition, 2001.

[Schlieder, 1993] C. Schlieder. Representing visible locations for qualitative navigation. pages 523–532. 1993.

[Schlieder, 1996] C. Schlieder. Ordering information and symbolic projection, 1996.

[Smith *et al.*, 1998] I. Smith, P. Cohen, J. Bradshaw, M. Greaves, and H. Holmback. Designing conversation policies using joint intention theory, 1998.

[Yeap and Jefferies, 1999] Wai K. Yeap and Margaret E. Jefferies. Computing a representation of the local environment. *Artificial Intelligence*, 107(2), 1999.

[Zheng and Tsuji, 1992] Jiang Yu Zheng and Saburo Tsuji. Panoramic representation for route recognition by a mobile robot. *nternational Journal of Computer Vision*, 9(1):55–76, 1992.

.

A Humanoid Listens to three simultaneous talkers by Integrating Active Audition and Face Recognition

Kazuhiro Nakadai†**, Daisuke Matsuura**¶ **Hiroshi G. Okuno**§**, and Hiroaki Kitano** ‡

† Honda Research Institute Japan Co., Ltd.
8-1 Honcho, Wako-shi, Saitama, 351-0114, Japan

¶ Graduate School of Science and Engineering, Tokyo Institute of Technology,
§ Graduate School of Informatics, Kyoto University,
‡ Kitano Symbiotic Systems Project, ERATO, Japan Science and Technology Corp.
nakadai@jp.honda-ri.com, matsuurd@mep.titech.ac.jp, okuno@nue.org, kitano@symbio.jst.go.jp

Content Areas: robotics, human computer interaction, speech processing, perception

Abstract

Humanoid robots may encounter situations where the number of simultaneous talkers exceeds that of its microphones. This paper addresses listening to three simultaneous talkers by a humanoid with two microphones. This task is difficult for human according to psychophysical observations. It is also difficult for automatic speech recognition systems (ASR), because the signal-to-noise ratio is quite low (around -3 dB) and noise is not stable due to interfering voices. Humanoid audition system consists of sound source separation, face recognition and ASR. Sound source separation is realized by an active direction-pass filter (ADPF), which extracts a sound source from a specified direction in real-time. Since features of sounds separated by ADPF vary according to the sound source direction, ASR uses multiple direction-dependent and speaker-dependent acoustic models. Among results of ASR with each acoustic model, the best one is selected by integrating the directional information by ADPF and speaker information by face recognition as well as a confidence measure of ASR results. The resulting system reduces error rates of ASR and thus is superior to human listeners in recognition of three simultaneous utterances, where three talkers were located 1 meter from the humanoid and apart from each other by 20 to 90 degrees at 10-degree intervals.

1 Introduction

Fifty years ago, Cherry [1953] discovered *cocktail party effect* which means that at a crowded party one can attend one conversation and then switch to another one. The essence of cocktail party effect resides in selective attention to one sound stream from an input mixture of sounds. By elaborating upon the cocktail party effect, one may expect that people or robots may listen to several utterances simultaneously. This is a dream for human, although a Japanese legendary says that Prince Shotoku who lived about 1,300 years ago could listen to ten people's petitions and give appropriate decisions at the same time. Psychophysical study showed that people can listen to at most two things at the same time [Kashino and Hirahara, 1996].

Research on computational auditory scene analysis (CASA) focuses on the computer modeling and implementation for the understanding of acoustic events [Rosenthal and Okuno, 1998]. One common way for CASA is sound source separation by using auditory cues such as common onset, offset, harmonic structure, amplitude modulation, frequency modulation, formants, and sound source localization. Other approaches to sound source separation are based on signal processing and information theory; microphone arrays with beam-forming techniques [Saruwatari *et al.*, 1999], and independent component analysis or blind source separation [Murata and Ikeda, 1998]. The problem of latter approaches is the theoretical limitation that the number of sound sources should be equal to or less than that of microphones.

Nakadai *et al.* [2001; 2002] developed an active direction-pass filter (ADPF) that separates sound source originating from a specified direction. The ADPF is installed in a humanoid robot called SIG, which can track multiple talkers in real-time. In this paper, we present the extension of the ADPF to separate sound sources and interfacing of the ADPF with automatic speech recognition (ASR) systems. The paper is organized as follows: In Section 2, related work is discussed. Section 3 proposes a system for ASR by integration of active audition and face recognition. Section 4 evaluates the system. Last two sections give future work and conclusion.

2 Related work

Some works reported that an application of sound source separation to front-end processing improved ASR under noisy environments [Park *et al.*, 1999; Yen and Zhao, 1996]. Nakagawa *et al.* [1999] reported on a challenging topic of separation and recognition of three simultaneous speeches with two

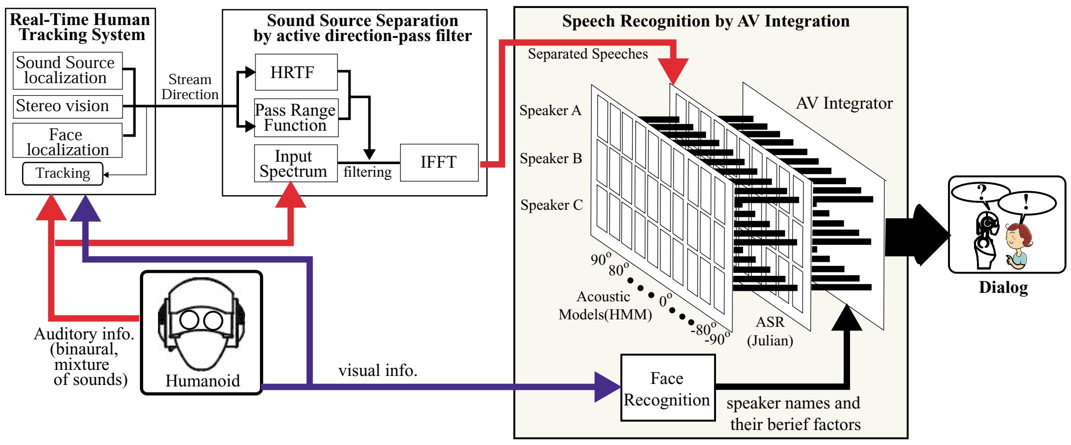


Figure 1: Speech Recognition System by Audio-Visual Integration

microphones by integrating visual and auditory information. They all use the binaural-based harmonic stream separation system (Bi-HBSS) that separates speech streams by using interaural phase difference (IPD) and interaural intensity difference (IID) obtained from a pair of microphones embedded in a dummy head in an anechoic room. The sound direction extracted by image processing is used to improve sound source separation, because the direction by vision is more accurate. This work, however, has studied with synthesized voices under simulated and off-line environment, and neither microphones nor sound sources are movable. Such assumptions are too strong to apply real-world systems such as mobile robots.

The difficulties in sound source separation and speech recognition under real world environments are discussed in robotic fields. Most robots that interact with people by ASR and gestures lack sound source separation function [Takanishi *et al.*, 1995; Breazeal and Scassellati, 1999; Matsusaka *et al.*, 1999]. Therefore, they have a microphone attached near the mouth of each speaker to avoid motor noise in motion and other sounds. Otherwise, they adopt the "stop-hear-act" principle; that is, a robot stops to hear. Asano *et al.* [2001] developed a robot which separates and recognizes sound sources by using a 8 ch circle microphone array in an ordinary office room. However, their system requires a lot of measurement for separation and localization in advance, and has difficulty in sound source separation during motion, while human can hear during motion. It is not enough for robot audition to be deployed in the real world yet.

From the viewpoint of improving ASR, various integration based approaches have been studied. An effective approach of such integration is the use of multiple results obtained from same or different ASRs. For example, ROVER that integrates different ASRs by a weighted voting method [Fiscus, 1997] and integration of ASRs based on confidence measure [Utsuro *et al.*, 2002] have been reported. Our approach improves ASR by integration of recognition results obtained from 51 ASRs processed in parallel corresponding to 51 direction- and speaker-dependent acoustic models. On the use of a large number of acoustic models, the integration of simple voting or majority rule often fails because a lot of missclassified results affect the system badly. Then, we integrate the results based on belief factor of word recognition. Another approach of improving ASR is audio-visual integration. It is efficient in ASR as well as in various fields such as sound source separation [Sodoyer *et al.*, 2002] and speaker recognition [Senior *et al.*, 1999]. Most of audio-visual integration in ASR use visual speech, that is, lip-reading [Luettin and Dupont, 1998; Silsbee and Bovik, 1996; Verma *et al.*, 1999]. In a robot, however, lip-reading is not always available because, when a person is away from the robot, resolution of images from robot's camera is insufficient for detecting his lips. His face is generally detected easier than the lips due to its size. Therefore, face recognition is more convenient for robots than lipreading. In this paper, improvement of ASR by integration of speech and face recognition is reported.

3 Simultaneous Speech Recognition by Audio-Visual Integration

The robot audition system for simultaneous talker recognition consists of a humanoid and three sub-systems – real-time human tracking, sound source separation by ADPF and speech recognition by audio-visual integration. The architecture of the system is shown in Fig. 1.

As a testbed of this work, the upper torso humanoid SIG is used. SIG has a cover by FRP (fiber reinforced plastic). It is designed to separate the SIG inner world from the external one acoustically. A pair of CCD cameras (Sony EVI-G20) is used for stereo vision. Two pairs of microphones are used for auditory processing. One pair is located in the left and right ear position for sound source localization. The other is installed inside the cover mainly for canceling self-motor noise in motion. SIG has 4 DC motors (4 DOFs) with functions of position and velocity control by using potentio-meters.

Sounds and images captured by SIG's microphones and cameras are sent to the real-time human tracking subsystem. The subsystem extracts directions of multiple sound sources from auditory and visual streams formed by fusing information obtained by sound source localization, multiple face localization and object localization by stereo vision [Nakadai *et* *al.*, 2001]. It also tracks one of the extracted sound sources according to focus-of-attention. The subsystem works in real time with a small latency of 200 ms by distributed processing with 5 PCs, networked through gigabit ether net.

The sound source directions are sent to the sound source separation by the ADPF. It separates sounds originating from the direction by using a pair of microphones [Nakadai *et al.*, 2002]. The filtering process is implemented by hypothesis matching for each sub-band of interaural intensity difference and interaural phase difference which are calculated from input spectra of left and right channels. The performance of the ADPF shows the difference of resolution in sound localization and separation. The resolution of localization and separation of the center of the humanoid is much higher than that of peripherals, indicating similar property of visual fovea (high resolution in the center of human eye). Therefore, we termed this phenomenon *auditory fovea*. To exploit this auditory fovea, the ADPF controls the pass range of the filter according to the sound direction and the direction of a head by motor movement. The improvement of about 9 dB in noise reduction is reported in separation of three simultaneous speech with the same loudness.

The speech recognition subsystem recognizes the extracted speech by multiple acoustic models and speaker name by face recognition.

3.1 Speech Recognition by AV Integration

The speech recognition subsystem consists of three processes. The first process is ASR by using multiple acoustic models. The acoustic models are direction- and speaker-dependent (DS-dependent). ASRs of which the number is the same as that of the acoustic models are processed in parallel. The second one is face detection and recognition, 3-best name list of a detected face and their belief factors are estimated. The last one is integration of speech and face recognition.

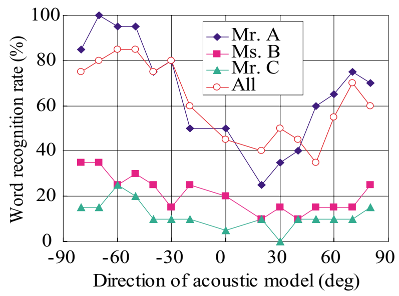
Speech Recognition by Multiple Acoustic Models

In the speech recognition subsystem, Hidden Markov Model (HMM) based acoustic models are used. The acoustic models are DS-dependent. The Japanese ASR software "Julian" is used for ASR. Multiple ASRs are processed in parallel, and all results are integrated in the integrator with results of the face recognition module.

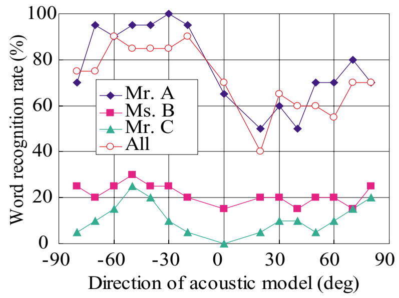
To make DS-dependent acoustic models, 150 words including numbers, colors and fruits by two men (Mr. A and Mr. C) and a woman (Ms. B) are used. Every word is played by loudspeakers of B&W Nautilus 805, and recorded by a pair of microphones installed in SIG. The loudspeakers and SIG are installed in a 3 m$\times$3 m room, the distance between each loudspeaker and SIG is 1 m shown in Fig. 3.

Three kinds of utterances are recorded as follows:

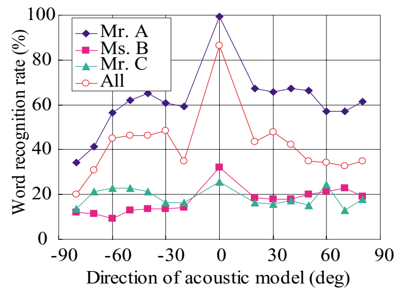
1. **single:** A loudspeaker is used for recording. The direction of the loudspeaker varies from -90° to 90° by 10° steps.
2. **double:** Two loudspeakers are used for recording simultaneously. The direction θ_2 of one loudspeaker is among 20°, 30°, $\cdots$, 80° and 90°. The direction of the other loudspeaker is 0° or $-\theta_2$.



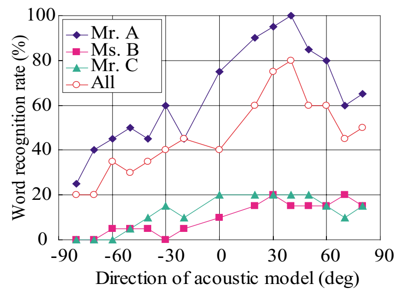
a) Mr. A, -60°



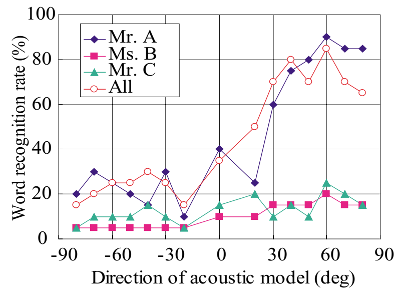
b) Mr. A, -30°



c) Mr. A, 0°



d) Mr. A, 30°



e) Mr. A, 60°

Figure 2: word recognition rates based on DS-dependent acoustic models (closed test)

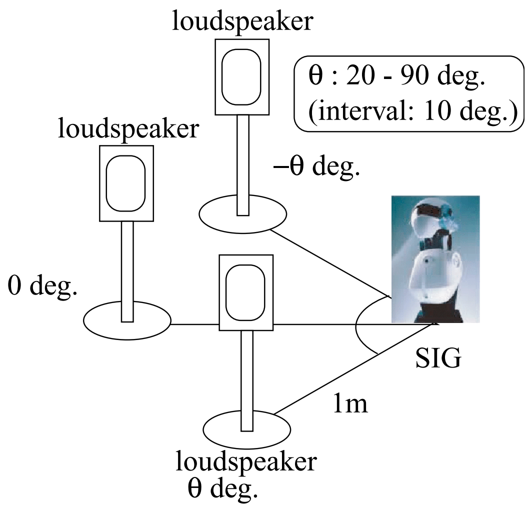


Figure 3: Recording Situation

3. **triple:** Three loudspeakers are used for recording simultaneously. The direction of the first loudspeaker is fixed to 0°. The direction θ_3 of the second loudspeaker is among 20°, 30°, $\cdots$, 80° and 90°. The direction of the last loudspeaker is $-\theta_3$.

To create training datasets for acoustic models, each speech is separated from recorded data (single, double and triple) by the ADPF under the condition that the directions of loudspeakers are given. The separated speeches are clustered by speaker and direction. As a result, 51 data sets (17 directions $\times$ 3 speakers) are obtained as training datasets. By using these training datasets, 51 acoustic models are trained. In this paper, each acoustic model is a triphone model trained 10 times by using Hidden Markov Model Toolkit.

Julian generates a score which represents logarithmic likelihood of the result. Each score is transformed to a belief factor P_s by using probability density function. Since the subsystem creates 51 results per input, 51 recognition results with belief factors are sent to the AV integrator.

Face Recognition

Since the visual processing detects several faces simultaneously, extracts, identifies and tracks each face, the size, direction, and brightness of each face changes frequently. The key idea is the combination of skin-color extraction, correlation based matching, and multiple scale images generation [Hidai *et al.*, 2000]. The face recognition module (see Fig. 1) projects each extracted face into the discrimination space, and calculates its distance to each registered face. Since this distance depends on the degree (the number of registered faces) of discrimination space, it is converted to a parameter-independent belief factor P_v by using probability density function. The discrimination matrix is created in advance or on demand using a set of variation of the face with a name. This analysis is done by Online Linear Discriminant Analysis [Hiraoka *et al.*, 2000].

Finally, the face recognition module sends 3-best face name with its belief factor P_v to the AV integrator.

Integration of speech and face recognition

The AV integrator receives 51 speech recognition results with belief factors and face name with a belief factor, and integrate them to output the most reliable result.

We tried integration of speech and face recognition by utilizing majority rule and voting such as ROVER [Fiscus, 1997]. Such integration method was effective only when the number of sound directions is at most three, that is, the number of DS-dependent acoustic models is around ten. The number of missclassified results increase, as the number of sound direction is large. Because a set of such wrong and same results affect the integration badly, the effectiveness of the integration based on majority rule or voting is weak in the situation where 17 sound directions are assumed.

To define a suitable algorithm for the integration of a large number of acoustic models, we measured word recognition rate against DS-dependent acoustic models when speaker and direction of input speech are fixed. Figures 2a) - e) are distributions of the results against Mr. A's speech from -60°, -30°, 0°, 30° and 60°, respectively. In these Figs, the x axis is direction of acoustic model, and the y axis is word recognition rates. The same speech data as a training dataset is used for recognition. The lines labeled "Mr. A", "Ms. B", "Mr. C" are the results by using acoustic models of "Mr. A", "Ms. B", "Mr. C". The line labeled "All" means the results by direction-dependent and speaker-independent acoustic model. These results show that the influence by direction is less than by speaker. When both of the person and the direction are correct, the word recognition rate is more than 90%, and better than that using speaker independent acoustic models. By taking the results in Fig. 2 into account, the AV integrator uses a cost function by Eq. 1 to integrate the results.

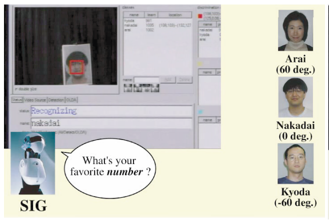
$$\begin{aligned} V(p_e) \;=\; \Bigg(&\sum_d r(p_e,d)\cdot v(p_e,d)\cdot P_s(p_e,d) \\ &+\sum_p r(p,d_e)\cdot v(p,d_e)\cdot P_s(p,d_e) \\ &-r(p_e,d_e)\cdot P_s(p_e,d_e)\Bigg)\cdot P_v(p_e). \end{aligned} \quad (1)$$

$$v(p,d)=\begin{cases} 1 & if \quad Res(p,d)=Res(p_e,d_e), \\ 0 & if \quad Res(p,d)\neq Res(p_e,d_e). \end{cases}$$

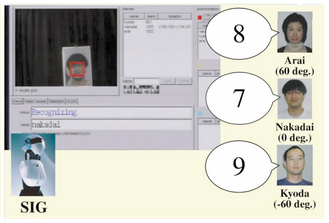
where $r(p,d)$ and $Res(p,d)$ are recognition rate shown in Fig. 2 and recognition result against input speech when an acoustic model of person p and sound direction d is used. The d_e is the sound source direction estimated by the real-time tracking system, and the p_e is a person to be evaluated. $P_v(p_e)$ is a probability in the face recognition module, and it is set to 0.5 when face recognition is unavailable. Finally, the AV integrator selects person p_e and result $Res(p_e,d_e)$ with the largest $V(p_e)$.

If the largest $V(p_e)$ is too small (less than 1) or close to the second largest one, SIG turns to the sound source and asks the person corresponding to the sound source again to make sure what he/she said.

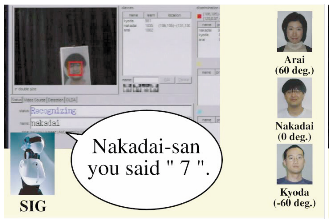
Thus, the system can recognize simultaneous speech and who spoke each speech by using multiple acoustic models and face recognition. The basic performance of the whole system is shown in Fig.4.



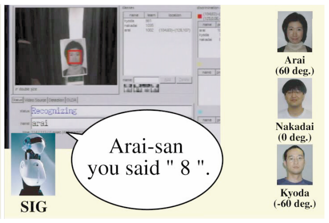
a) SIG asks a question.



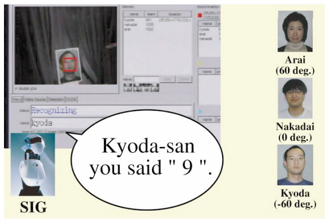
b) Everyone replies simultaneously.



c) SIG answers what Mr. A said.



d) SIG answers what Ms. B said.



e) SIG answers what Mr. C said.

Figure 4: Snapshots of Three Simultaneous Speech Recognition

4 Evaluation

The system is evaluated through four kinds of "three" simultaneous speech recognition as follows:

Exp. 1: a scenario of a simple human-humanoid interaction,
Exp. 2: word recognition without using face recognition,
Exp. 3: word recognition by audio-visual integration, and
Exp. 4: three simultaneous speech recognition by human.

In every experiment, room conditions and locations of loudspeaker and SIG are the same as those described in Sec. 3.1. The three loud speakers are attached photographs of speakers for face recognition instead of real humans. For recording of three simultaneous speech, a three-word combination is selected from a list of three-word combinations in training datasets. Then, three loudspeakers play the three words according to the combination. The mixture of sounds is captured by SIG's microphones and sent to the system.

The first experiment shows the basic performance of the system. The scenario is as follows:

1. SIG asks three persons (Mr. A, Ms. B and Mr. C) about the favorite number.
2. Each speaker selects any of 1 to 10, and they speak simultaneously.
3. SIG separates and recognizes the mixture of speeches. SIG also identifies who spoke the number.
4. When SIG fails speech recognition or speaker identification, it turns to the speaker and asks the question again.
5. Finally, SIG replies the numbers with person's names.

In the initial situation of the scenario, the loudspeaker *A* attached Mr. A's photograph is located at the front direction of SIG. The loudspeaker *B* attached Ms. B's photograph and the loudspeaker *C* attached Mr. C's are located at the 60° and -60°, respectively. A typical result is observed as follows:

1. SIG asks a question about the favorite number in Fig. 4a).
2. The speakers play three words simultaneously in Fig. 4b).
3. SIG localizes the speakers by using the real-time tracking subsystem. The speech from the obtained direction is extracted by the sound source separation subsystem. Each separated sound is recognized by using multiple acoustic models. The integrator integrates the multiple results of speech recognition with speaker names obtained by face recognition, and decides the best result of each separated sound.
4. SIG answers the numbers with the names of speakers in Fig. 4c)-e).

In Exp. 2 - 4, the direction of first speaker is fixed to 0 $\circ$. The second speaker direction θ varies from 20° to 90° by 10°. The direction of the last loudspeaker is $-\theta$. Figures 5 and 6 show word recognition rate without and with face recognition, respectively. In Exp. 2, P_v in Eq.1 is fixed to 0.5 because face recognition is unavailable. The X and Y axes of figures mean direction difference between loudspeakers θ and word recognition rate in percentage. The lines labeled "left", "center" and "right" are 1-best recognition rates of left, center and right loudspeakers, respectively. The dotted lines are 3-best recognition rates.

Figure 7 shows word recognition rate by a testee. During the measurement, the direction of the testee's head is fixed to 0°. The dotted lines labeled "left", "center" and "right" are recognition rates of left, center and right loudspeakers when the testee listens to simultaneous speech once. The lines are recognition rates when the testee listens to the same simultaneous speech three times.

Figures 5 and 6 prove the efficiency of the audio-visual integration. The changes of word recognition rate against direction difference between loudspeakers are smaller in Fig. 6. This indicates that the audio-visual integration compensates such changes. In Fig. 6, the recognition rates of 1-best and 3-best are close. This means that the Aaudio-visual integration encourages a belief factor of a correct answer properly. In comparison with Fig. 7, the audio-visual integration provides better performance than human speech recognition in one time listening (dotted lines). It may be unfair to compare with one time listening, because multiple ASRs are processed in parallel in Fig. 6. It can be said that the results of three time listening in Fig. 7 correspond to the 1-best results in Fig. 6 because he can focus on a loudspeaker per a trial. The recognition rate difference between the audio-visual integration and three time listening is about 10%.

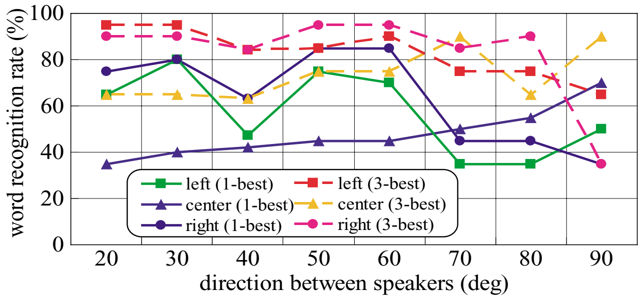


Figure 5: word recognition rate (baseline)

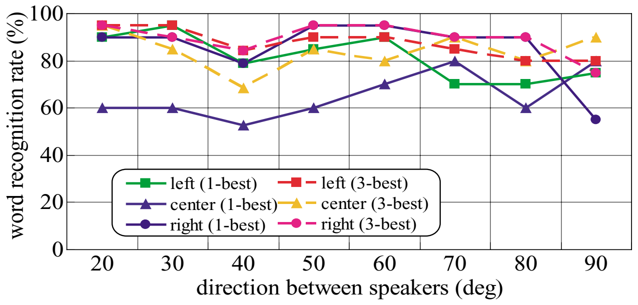


Figure 6: improvement of word recognition by audio-visual integration

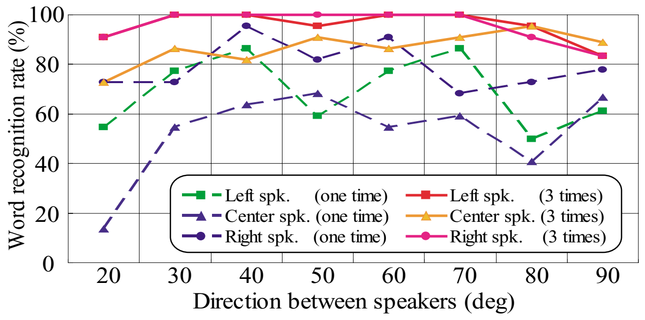


Figure 7: word recognition by human

In human, the word recognition rate improves more than 20% by listening three times in Fig. 7. Although he does not move his head, this means that he can change the directivity of his audition. This is a kind of active audition in human.

Figures 5 and 6 shows that the performance of the front loudspeaker is worse than side ones. The same tendency is found in Fig. 7. The reason of this phenomenon is that both the left and the right loudspeakers affect sound from the front one, while sound from left and right loudspeakers is distorted by only the front one, because the recognition rate improves as the direction difference between loudspeakers is larger. In simulated environments, Nakagawa *et al.* [1999] reported the similar tendency. The 3-best result in the audio-visual integration resolves the bad influence from side loudspeakers.

From the viewpoint of sound source separation by the ADPF, sound from the front direction is extracted better than that of side direction. Therefore, speech recognition rate of the front direction is expected to be the best under the condition that influence from other sound sources is constant. This evaluation by using 5 loud speakers is a future work.

In Figs. 5 and 6, ASR of the left and the right loud speakers is getting worse as leaving the front direction. We consider that this is caused by directivity of microphones in SIG. The microphones are omni-directional, but the microphone has directivity of the front direction by the cover of SIG. In this case, active motion to turn to the sound source improves ASR. In addition, when a face is detected by such turning, the audio-visual integration improves ASR.

5 Future Work

The system uses a general HMM based ASR engine. However, we should consider characteristics of front-end processing in ASR. The ADPF separates sound sources based on sub-band selection in frequency domain. Therefore, some sub-bands can be dropped by separation errors. In this case, ASR engine should be modified to cope with such missing sub-bands by introducing missing feature theory [Tibrewala and Hermansky, 1997].

In the real world, a sudden and loud noise affects across wide frequency ranges. In this case, signals at the time when the noise happens cannot be used for ASR. To cope with such missing data, missing data theory [Renevey *et al.*, 2001; Barker *et al.*, 2001] should be introduced.

In this paper, we assume known speakers and isolated word recognition. When an unknown speaker exists, speaker independent acoustic models are necessary for unknown speakers. Such a mechanism should be introduced to the system. The word spotting techniques are effective for speech recognition of longer sentences.

6 Conclusion

Three simultaneous speech recognition by two microphones is described. The results show that various kinds of integration – use of multiple DS-dependent acoustic models, audio-visual integration by combination of face recognition and DS-dependent acoustic models, and active audition combining audition with motion – is efficient and essential to improve ASR. In the integration of a large number of results, simple voting or majority rule are of less use. The system proves that our integration based on belief factor is effective. The system described in this paper mainly aims to realize user interface for human-humanoid interaction. However, the techniques introduced to the system can be applied to any system with microphones and camera. Therefore, we can expect various applications such as TV conference system, automatic shorthand and super directional microphones by their deployment to the real-world.

Acknowledgments

We thank Prof. Hiroshi Mizoguchi, Tokyo Univ. of Science for his valuable discussions. We also thank Chihaya Okuno, Yuusuke Tanemura, Kouji Kyoda and Michiru Arai for their help.

References

[Asano *et al.*, 2001] F. Asano, M. Goto, K. Itou, and H. Asoh. Real-time sound source localization and separation system and its application to automatic speech recognition. In *Proceedings of International Conference on Speech Processing (Eurospeech 2001)*, pages 1013–1016. ESCA, 2001.

[Barker *et al.*, 2001] J. Barker, M.Cooke, and P.Green. Robust asr based on clean speech models: An evaluation of missing data techniques for connected digit recognition in noise. In *Proc. of 7th European Conference on Speech Communication Technology (EUROSPEECH-01)*, volume 1, pages 213–216. ESCA, 2001.

[Breazeal and Scassellati, 1999] C. Breazeal and B. Scassellati. A context-dependent attention system for a social robot. In *Proc. of the Sixteenth International Joint Conference on Atificial Intelligence (IJCAI-99)*, pages 1146–1151, 1999.

[Cherry, 1953] E. C. Cherry. Some experiments on the recognition of speech, with one and with two ears. *Journal of Acoustic Society of America*, 25:975–979, 1953.

[Fiscus, 1997] J.G. Fiscus. A post-processing systems to yield reduced word error rates: Recognizer output voting error reduction (rover). In *Proceedings of the Workshop on Automatic Speech Recognition and Understanding (ASRU-97)*, pages 347–354. IEEE, 1997.

[Hidai *et al.*, 2000] K. Hidai, H. Mizoguchi, K. Hiraoka, M. Tanaka, T. Shigehara, and T. Mishima. Robust face detection against brightness fluctuation and size variation. In *Proc. of IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS-2000)*, pages 1397–1384. IEEE, 2000.

[Hiraoka *et al.*, 2000] K. Hiraoka, M. Hamahira, K. Hidai, H. Mizoguchi, T. Mishima, and S. Yoshizawa. Fast algorithm for online linear discriminant analysis. In *Proceedings of ITC-2000*, pages 274–277. IEEK/IEICE, 2000.

[Kashino and Hirahara, 1996] M. Kashino and T. Hirahara. One, two, many – judging the number of concurrent talkers. *Journal of Acoustic Society of America*, 99(4):Pt.2, 2596, 1996.

[Luettin and Dupont, 1998] J. Luettin and S. Dupont. Continuous audio-visual speech recognition. In *Proceeding of 5th European Conference on Computer Vision (ECCV-98)*, volume II of *Lecture Notes in Computer Science*, pages 657–673. Springer Verlag, 1998. IDIAP-RR 98-02.

[Matsusaka *et al.*, 1999] Y. Matsusaka, T. Tojo, S. Kuota, K. Furukawa, D. Tamiya, K. Hayata, Y. Nakano, and T. Kobayashi. Multi-person conversation via multi-modal interface — a robot who communicates with multi-user. In *Proc. of 6th European Conference on Speech Communication Technology (EUROSPEECH-99)*, pages 1723–1726. ESCA, 1999.

[Murata and Ikeda, 1998] N. Murata and S. Ikeda. An online algorithm for blind source separation on speech signals. In *Proceedings of 1998 International Symposium on Nonlinear Theory and its Applications*, pages 923–927, 1998.

[Nakadai *et al.*, 2001] K. Nakadai, K. Hidai, H. Mizoguchi, H. G. Okuno, and H. Kitano. Real-time auditory and visual multiple-object tracking for robots. In *Proc. of the 17th Int. Joint Conf. on Atificial Intelligence (IJCAI-01)*, pages 1424–1432. MIT Press, 2001.

[Nakadai *et al.*, 2002] K. Nakadai, H. G. Okuno, and H. Kitano. Exploiting auditory fovea in humanoid-human interaction. In *Proceedings of 18th National Conference on Artificial Intelligence (AAAI-2002)*, pages 431–438. AAAI, 2002.

[Nakagawa *et al.*, 1999] Y. Nakagawa, H. G. Okuno, and H. Kitano. Using vision to improve sound source separation. In *Proc. of the 16th National Conference on Artificial Intelligence (AAAI-99)*, pages 768–775. AAAI, 1999.

[Park *et al.*, 1999] H. M. Park, H. Y. Jung, T. W. Lee, and S. Y. Lee. Subband-based blind signal separation for noisy speech recognition. *Electronics Letter*, 35(23):2011–2012, 1999.

[Renevey *et al.*, 2001] Philippe Renevey, Rolf Vetter, and Jens Kraus. Robust speech recognition using missing feature theory and vector quantization. In *Proc. of 7th European Conference on Speech Communication Technology (EUROSPEECH-01)*, volume 2, pages 1107–1110. ESCA, 2001.

[Rosenthal and Okuno, 1998] D. Rosenthal and H. G. Okuno, editors. *Computational Auditory Scene Analysis*. Lawrence Erlbaum Associates, Mahwah, New Jersey, 1998.

[Saruwatari *et al.*, 1999] H. Saruwatari, S. Kajita, K. Takeda, and F. Itakura. Speech enhancement using nonlinear microphone array based on complementary beamforming. *IEICE Trans. fundamentals*, E82-A(8), 1999.

[Senior *et al.*, 1999] A. Senior, C.V. Neti, and B. Maison. On the use of visual information for improving audio-based speaker recognition. In *Proceeding of Audio-Visual Speech Processing (AVSP-99)*. ESCA, 1999.

[Silsbee and Bovik, 1996] P.L. Silsbee and A.C. Bovik. Computer lipreading for improved accuracy in automatic speech recognition. *IEEE Transactions on Speech and Audio Processing*, 4(5):337–351, 1996.

[Sodoyer *et al.*, 2002] D. Sodoyer, L. Girin, C. Jutten, and J. L. Schwartz. Audio-visual speech sources separation – a new approach exploiting the audio-visual coherence of speech stimuli –. In *Proceeding of the Internationa l Conference on Spoken Language Processing (ICSLP-2002)*, pages 1953–1956. ISCA, 2002.

[Takanishi *et al.*, 1995] A. Takanishi, S. Masukawa, Y. Mori, and T. Ogawa. Development of an anthropomorphic auditory robot that localizes a sound direction (*in japanese*). *Bulletin of the Centre for Informatics*, 20:24–32, 1995.

[Tibrewala and Hermansky, 1997] S. Tibrewala and H. Hermansky. Sub-band based recognition of noisy speech. In *Proceeding of IEEE International Conference on Acoustic, Speech, and Signal Processing (ICASSP-1997)*, pages 1255–1258. IEEE, 1997.

[Utsuro *et al.*, 2002] T. Utsuro, T. Harada, H. Nishizaki, and S. Nakagawa. A confidence measure based on agreement among multple lvcsr models -correlation between pair of acoustic models and confederence-. In *Proceeding of the International Conference on Spoken Language Processing (ICSLP-2002)*, volume 1, pages 701–704. ISCA, 2002.

[Verma *et al.*, 1999] A. Verma, T. Faruquie, C. Neti, and S. Basu. Late integration in audio-visual continuous speech recognition. In *Proceeding of the Workshop on Automatic Speech Recognition and Understanding (ASUR-1999)*. IEEE, 1999.

[Yen and Zhao, 1996] K. C. Yen and Y. Zhao. Robust automatic speech recognition using a multi-channel signal separation front-end. In *Proceeding of the International Conference on Spoken Language Processing (ICSLP-96)*, volume 3, pages 1337–1340. ISCA, 1996.